Fachhochschule Köln
Cologne University of Applied Sciences

Framework zur Integration von Usability Engineering in agile Softwareentwicklungsprozesse

Bachelorarbeit

ausgearbeitet von

René Rappenhöner

vorgelegt an der

Fachhochschule Köln
Campus Gummersbach
Fakultät für Informatik und
Ingenieurwissenschaften

im Studiengang

Medieninformatik

Erster Prüfer:	Prof. Dr. Gerhard Plaßmann Fachhochschule Köln
Zweiter Prüfer:	Prof. Dr. Mario Winter Fachhochschule Köln

Köln, im August 2009

Adressen: René Rappenhöner
Wilhelm-Marx-Straße 10
51067 Köln
r.rappenhoener@web.de

Prof. Dr. Gerhard Plaßmann
Fachhochschule Köln
Institut für Informatik
Steinmüllerallee 1
51643 Gummersbach
gerhard.plassmann@fh-koeln.de

Prof. Dr. Mario Winter
Fachhochschule Köln
Institut für Informatik
Steinmüllerallee 1
51643 Gummersbach
mario.winter@fh-koeln.de

Kurzfassung

Bei der Entwicklung interaktiver Software werden die Belange der Benutzer oft außer Acht gelassen oder nur unzureichend berücksichtigt. Dies führt dazu, dass das System von den Benutzern nicht akzeptiert wird, es die Benutzer nicht im ausreichenden Maße unterstützt oder, im Ernstfall, die Benutzer ihre Aufgaben nicht vollständig lösen können. Um einen Nährboden für die Entwicklungsprozesse von gebrauchstauglicher Software zu schaffen, muss eine grundlegende Entscheidung getroffen werden. Der Usability Engineering Prozess wird selbst geplant und durchgeführt, oder extern an eine Firma weitergegeben. Diese Arbeit beschäftigt sich mit ersterem und integriert den Usability Engineering- und Softwareentwicklungsprozess in einem Framework mit agilem Vorgehensmodell, welches sich an erfahrene Projektleiter, Projektmanager und Softwareentwickler ohne gefestigtes Wissen im Bereich des Usability Engineerings richtet. Es werden Ziele definiert und mögliche Methoden und Techniken vorgestellt, mit denen diese Ziele erreicht werden können. Da sich das Framework an Softwareentwickler richtet, die einen Usability Enginigeering Prozess integrieren wollen, werden zwar notwendige Ziele der Softwareentwicklung, aber nicht die Methoden und Techniken diese zu erreichen, vorgegeben, um einen etablierten Softwareentwicklungsprozess nicht zu sehr zu beeinflussen.

Inhaltsverzeichnis

1 Motivation, Problem- und Aufgabenstellung 6
1.1 Ist-Situation . . . 6
1.2 Problem . . . 7
1.3 Lösungsansatz . . . 8

2 Psychologische Grundlagen 10
2.1 Gedächtnis . . . 10
2.1.1 Sensorischer Speicher . . . 11
2.1.2 Arbeitsgedächtnis . . . 11
2.1.3 Langzeitgedächtnis . . . 11
2.1.4 Zusammenarbeit der Gedächtnisse . . . 12
2.2 Schemata . . . 13
2.3 Mentale Modelle . . . 14
2.4 Wahrnehmung . . . 14
2.4.1 Visuelle Wahrnehmung (*feature integration theory*) . . . 14
2.4.2 Erkenntnisse aus der Gestaltpsychologie . . . 15
2.4.3 Goldener Schnitt . . . 17

3 Einführung in relevante Standards 18
3.1 ISO 9241 . . . 18
3.2 ISO 13407 . . . 19
3.3 ISO/PAS 18152 . . . 21
3.4 ISO 12207 . . . 24
3.4.1 Begriffsdefinition . . . 24
3.4.2 Primärprozesse . . . 25
3.4.3 Unterstützende Prozesse . . . 25
3.4.4 Organisatorische Prozesse . . . 26
3.4.5 Entwicklungsprozess . . . 27

4 Integrations-Möglichkeiten der Standards 35

5 Scrum 38
5.1 Rollen . . . 40
5.1.1 Product Owner . . . 40
5.1.2 Scrum Master . . . 40
5.1.3 Team . . . 40
5.2 Prozesse . . . 41
5.2.1 Product Backlog erstellen . . . 41
5.2.2 Sprint Backlog erstellen . . . 42

5.2.3 Daily-Scrum . . . 42
5.2.4 Sprint Review . . . 42
5.2.5 Sprint Retrospektive . . . 43

6 Integration der Standards in Scrum 44

7 Framework 45
7.1 Begriffsklärung . . . 46
7.2 Vorgehensmodell . . . 47
7.3 Integration der ISO 13407 . . . 51
7.3.1 Notwendigkeit von benutzerzentrierter Entwicklung erkennen . . . 51
7.3.2 Kontextanalyse . . . 51
7.3.3 Anforderungsanalyse . . . 52
7.3.4 Gestaltungslösungen entwickeln . . . 53
7.3.5 Gestaltungslösungen evaluieren . . . 54
7.3.6 Komponenten entwickeln . . . 54
7.3.7 Komponenten evaluieren . . . 55
7.4 Methoden und Techniken . . . 55
7.4.1 Grundlegendes . . . 55
7.4.2 Kontext- und Anforderungsanalayse . . . 56
7.4.3 Design und Prototyping . . . 63
7.4.4 Evaluation und Testing . . . 75

8 Fazit 81

9 Ausblick 82

Abbildungsverzeichnis 83

Tabellenverzeichnis 84

Glossar 86

Literaturverzeichnis 87

Eidesstattliche Erklärung 88

1 Motivation, Problem- und Aufgabenstellung

1.1 Ist-Situation

Die Firma OPITZ CONSULTING ist ein erfolgreiches Unternehmen, welches seit der Gründung 1990, stetig wächst und hauptsächlich im Bereich des Business Engineering und der Softwareentwicklung tätig ist. Auf dem Gebiet der Datenbank- und Softwareentwicklung genießt die Firma einen guten Ruf. Die meisten Erfahrungen liegen dort bei Oracle Datenbanken, wobei andere Systeme auch übernommen und gewartet werden. Software wird in der Regel in Java entwickelt, wobei auch Systeme gewartet werden, die in .Net oder älteren Programmiersprachen entwickelt wurden.
OPITZ CONSULTING versteht sich selbst als Unternehmen das Organisationsberatung und IT-Projektabwicklung vereint. Es wird immer eine partnerschaftliche Beziehung zum Kunden angestrebt, so dass beide Seiten offen über Probleme reden können, die im Projektverlauf entstehen. Ein ebenso wichtiger Faktor ist die Qualität der Lösungen. Es ist dem Unternehmen enorm wichtig, die beste Lösung für den Kunden zu finden, da sie nicht nur eine Architektur verkaufen wollen, die gerade in Mode ist. Große Kunden sind beispielsweise

- Postbank
- WestLB
- Eplus
- Thyssen Krupp Steel
- VDI
- RWE
- INTERSEROH
- ITENOS

Es gibt viele Expertisen in verschiedenen Domänen. Auf Konferenzen wie der JAX[1] oder der DOAG[2] werden regelmäßig Vorträge von OPITZ CONSULTING Mitarbeitern gehalten. Generell sind Veröffentlichungen gerne gesehen und werden vom Unternehmen unterstützt. OPITZ CONSULTING bietet auch Schulungen für interne (als Unterstützung der Weiterbildung) und externe Personen an. Beispielsweise SOA-, Oracle- oder Java-Schulungen mit diversen Schwerpunkten.

1.2 Problem

Wie eine vorrausgegange Analyse [Ren09] gezeigt hat, gibt es Defizite im Bereich des Usability Engineerings. Im Rahmen der Analyse wurde herausgefunden, dass gerade im Bereich des Nutzungskontextes (nach ISO/PAS 18152) [3] erhebliche Probleme zu finden sind. Es gibt keine Benutzerbeschreibungen, Beschreibung der Arbeitsumgebung oder dokumentierte Benutzeraufgaben. Dies bedeutet, dass ohne deskriptives Modell ein präskriptives Modell, Software also ohne konkrete Situationsbeschreibung, entwickelt wird. Durch diesen Umstand ist das Ergebnis der Evaluation der Software unvollständig, da ohne konkrete Ziele, welche sich auch aus einer Analyse eines deskriptivem Modells ergeben, keine Evaluation durchgeführt werden kann. Ohne deskriptive und präskriptive Aufgabenmodelle kann keine aufgabenbezogene Evaluation durchgeführt werden. In der ISO 9241 wird die Gebrauchstauglichkeit als das Ausmaß, in dem ein bestimmtes Produkt durch bestimmte Benutzer in einem bestimmten Nutzungskontext genutzt werden kann, um bestimmte Ziele effektiv, effizient und zufriedenstellend erreichen kann, definiert. Wenn die Benutzer und die Aufgaben, die sie effizient, effektiv und zufriedenstellend lösen wollen, nicht beschrieben sind, können folglich die Effizienz, Effektivität oder die Zufriedenheit nicht gemessen und so die Gebrauchstauglichkeit nicht geprüft werden.
Da bein OPITZ CONSULTING keine Usability-Experten vorhanden sind, sondern „nur“ Softwareentwickler, Projektleiter und Projektmanager die keine bis wenig Erfahrung im Bereich des Usability Engineering haben, sind klar definierte Prozesse erforderlich, bei denen auch besonders gut dokumentiert sein sollte, wieso diese erforderlich und was die Ziele der Prozesse sind und wie diese erreicht und geprüft werden können.

[1] Die JAX ist eine Konferenz für Java, Enterprise Architekturen und SOA
[2] Die Deutsche Oracle Anwendergruppe veranstaltet regelmäßige Treffen, auf denen aktuelle Themen vorgestellt und diskutiert werden.
[3] In der Analyse wurde der Entwicklungsprozess von OPITZ CONSULTING auf die geforderte Basispraktikten der ISO/PAS 18152 geprüft. Für den Erfüllungsgrad dieser Basispraktiker wurde das Bewertungsmodell der ISO 15504 verwendet.

1.3 Lösungsansatz

Ziel der Arbeit ist es, ein Usability Framework für agile Softwareentwicklungsprozesse zu entwickeln, welches nicht von Usability Experten, sondern von Softwareentwicklern durchgeführt werden kann. Das Framework soll sich nach dem Vorgehen der ISO 13407 richten und den geforderten Basispraktikten der ISO/PAS 18152 gerecht werden. Da in der ISO/PAS 18152 nur Ziele definiert werden, soll den Entwicklern Methoden und Techniken zur Verfügung gestellt werden, die sie unterstützt diese zu erreichen. Diese Methoden und Techniken sollen so erläutert werden, dass sie von erfahrenen Softwareentwicklern, Projektleitern und Projektmanagern, die allerdings keine Erfahrungen mit Usability-Engineering-Prozessen besitzen, durchgeführt werden können. Dafür müssen die Ziele klar definiert und Möglichkeiten aufgezeigt werden, die Zielerreichung zu prüfen. Ebenso müssen die einzelnen Methoden und Techniken detailliert beschrieben und deren Sinn erläutert werden, um ein Bewusstsein für benutzerzentrierte Entwicklung zu schaffen und die Projektmitarbeiter in die Lage zu versetzen, ein fachlich korrektes Usability-Engineering durchzuführen. Hierfür sollen die Projektmitarbeiter auch in die Lage versetzt werden, angemessene Methoden und Techniken für den jeweiligen Prozess auszuwählen. Hierdurch soll die Prozessqualität gesteigert und eine höhere Softwarequalität, bezüglich der Gebrauchstauglichkeit, erreicht werden. Die eigentliche Entwicklung der Software soll nicht beeinflusst werden, es geht vielmehr darum Integrationspunkte zu finden und über diese eine umfassendere Entwicklung zu schaffen. Die Integrationspunkte sollen über die Aktivitäten, die in beiden Vorgehensweisen ähnlich sind, jedoch aus verschiedenen Perspektiven betrachtet werden, stattfinden. Neben den einzelnen Prozessen sollen auch generelle Verhaltensregeln bei der Entwicklung erläutert werden, wie beispielsweise intensive Kommunikation zwischen den Entwicklern untereinander und mit den Benutzern.
Für das Framework soll ein Vorgehensmodell entwickelt werden, was geeignet ist vom Projektmanagement erfasst, gemanaged und kontrolliert zu werden. Mögliche Softwarelösungen für die Entwicklung des Vorgehensmodells sind VModell XT Editor, BM Rational Method Composer oder MicroTOOL InStep.
Das entwickelte Framework soll sich ausschließlich an Scrum-Projekte richten, da es zeitlich nicht möglich ist, weitere agile Vorgehensweisen zu betrachten. In Scrum kann formativ evaluiert und in Etappen entwickelt werden. So besteht die Möglichkeit, Probleme technischer Natur, aber auch Probleme mit der Benutzbarkeit des Systems, frühzeitig zu erkennen und dementsprechend entgegen zu wirken. Es müssen Rollen, Aktivitäten und Artefakte identifiziert werden, die eine Kommunikation und Zusammenarbeit zwischen dem Usability-Engineering- und dem Softwareentwicklungsprozess ermöglichen. Ebenso müssen die Rollen, die ein Scrum Projekt vorgibt, analysiert und

deren Einsatzmöglichkeiten für das Framework im Usability Engineering und der Softwareentwicklung aufgezeigt werden.

Vorgehen

Damit das Framework verstanden und verwendet werden kann, müssen einige Informationen bereit gestellt werden. Nötige Informationen sind beispielsweise psychologische Grundlagen, um zu verstehen wie Benutzer denken und Handeln und wie die Kommunikation innerhalb des Entwicklungsteams sein sollte. Ebenso sind nationale und internationale Standards von Belang, die im Bereich der Softwareentwicklung und im Bereich des Usability Engineerings relevant sind. Scrum wird als agiles Vorgehensmodell, mit allen notwendigen Prozessen und Eigenschaften, vorgestellt, um eine Integration der Softwareentwicklung, des Usability Engineerings und der Ziele der Standards zu ermöglichen und in einem neuen, verfeinerten Vorgehensmodell zu vereinen.
Die Arbeit besteht daher aus zwei Teilen. Um ein besseres Verständnis des Frameworks zu erreichen, werden im ersten Teil nötige Informationen zu den einzelnen Prozessen, Aktivitäten, Artefakten, Standards und Integrationsmöglichkeiten bereitgestellt. Im zweiten Teil wird das Framework im Detail beschrieben und Möglichkeiten bereitgestellt, wie die Ziele der einzelnen Prozesse erreicht werden können.

2 Psychologische Grundlagen

Die psychologischen Erkenntnisse, die bei der Verwendung von interaktiven Systemen zum Tragen kommen, sind sehr vielfältig und wachsen stetig. Für das Verständnis mancher Methoden und Verhaltensweisen bei benutzerzentrierter Entwicklung, wurden hier einige wichtige Aspekte ausgewählt und in den folgenden Abschnitten beschrieben. Zu beachten ist, dass diese Ausführungen nur ein kleiner Teilausschnitt sind, da die Themen teilweise so komplex sind, dass man ein oder auch mehrere Bücher darüber schreiben könnte.
Aus der Kognitionspsychologie ist bekannt, dass menschliches Handeln und Problemlösen besonders dann effizient ist, wenn die aktivierten mentalen Strukturen zu den Strukturen des wahrgenommenen Problems passen (Vgl. [RLS04], S. 451 ff.). Die Arbeitsanalyse ist von grundlegender Bedeutung, um die Wissenstrukturen, die sich in den Arbeitsschritten der Benutzer widerspiegeln, zu erfassen. Durch diese Erkenntnisse können die Interaktion und das Interface so gestaltet werden, dass die Benutzer ihre vorhanden mentalen Modellen darauf anwenden können. Aus diesem Grund ist die Arbeitsanalyse von großer Bedeutung bei der Entwicklung gebrauchstauglicher Systeme. Die folgende Ausführungen sollen ein Basiswissen wie auch Grundverständnis schaffen und dazu beitragen, den Sinn der Methoden und Techniken zu verstehen, die in diesem Framework verwendet werden.

2.1 Gedächtnis

Hier wird ein horizontales Gedächtnismodell nach Atkinson und Shiffrin betrachtet. Neben den horizontalen Gedächtnismodellen, bei denen die Informationen seriell von verschiedenen Verarbeitungseinheiten verarbeitet werden und die Intensität der Verarbeitung die Tiefe der Speicherung beeinflusst, existieren vertikale Gedächtnismodelle, bei denen die Tiefe der Speicherung durch die unterschiedlichen Ebenen der Verarbeitung zustande kommen. Aus beiden Modellen können Informationen abgeleitet werden, die bei der Entwicklung interaktiver Systeme hilfreich sein können. Jedoch ist das horizontale Gedächtnismodell im Kontext dieser Arbeit von größerer Bedeutung, da gerade mit der Verarbeitung im Arbeitsgedächtnis einige Grundlagen der menschlichen Infor-

mationsverarbeitung und daraus resultierende Handlungen, wie auch Einschränkungen, erklärt werden können.

2.1.1 Sensorischer Speicher

Im sensorischen Speicher (*sensory store*), werden visuelle oder auch auditive Informationen temporär für das Arbeitsgedächtnis gespeichert. Die Verweildauer ist relativ gering und die Übermittlung ins Arbeitsgedächtnis findet in Sekundenbruchteilen statt.

2.1.2 Arbeitsgedächtnis

Im Arbeitsgedächtnis (*working memory*) werden Informationen verarbeitet. Im Gegensatz um Langzeitgedächtnis ist die Halbwertzeit von Informationen sehr gering, ca. 17 Sekunden. Da das Arbeitsgedächtnis, wie der Name schon sagt, aktiv beim lösen von Aufgaben beteiligt ist, sind die Kapazitäten der zu verarbeitenden Daten beschränkt. Nelson Cowan hat herausgefunden, dass die Anzahl der Informationseinheiten, sogenannte chunks, auf 4 ± 1 beschränkt sind[1]. Allerdings ist es möglich, chunks zu aggregieren. Beispielsweise sind Telefonnummern mit Vorwahl in der Regel länger als fünf Ziffern, können aber, wenn sie gerade gehört wurden, also nicht aus dem Langzeitgedächtnis kommen, behalten und wiedergegeben werden. Wenn man sich eine Telefonnummer mit den Ziffern 0 2 2 6 1 1 1 2 5 6 8 9 vorstellt, wirkt diese erst einmal sehr umfangreich, wenn man aber Chunks bildet, wird diese besser behalten. Für Jemanden aus der Umgebung von Gummersbach ist die Vorwahl 02261 vermutlich schon ein Chunk, da es sich um die Gummersbacher Vorwahl handelt, welche aus dem Langzeitgedächtnis geladen werden kann. Die Rufnummer kann nun verschieden aggregiert werden, man könnte sich 11 25 68 9 oder auch 112 568 9 merken. Für jemanden, der die Vorwahl nicht kennt, also keine Unterstützung aus dem Langzeitgedächtnis bekommt, ist das Behalten der Nummer weitaus schwieriger. Noch schwieriger wird es für Personen die aufgrund einer Schwäche oder Unkenntnis über mathematische Grundlagen nicht in der Lage sind, diese Ziffern zu aggregieren.

2.1.3 Langzeitgedächtnis

Im Langzeitgedächtnis (*long term memory*) werden Informationen gespeichert, die nicht nur kurzzeitig, sondern längerfristig benötigt werden. Informationen, die oft benötigt werden, können schneller und detaillierter abgerufen werden als solche, die selten be-

[1] Bis 2002 wurde aufgrund der Forschungsergebnisse von George Miller davon ausgegangen, dass 7 ± 2 chunks Möglich sind. Diese Aussage wird mittlerweile nicht mehr akzeptiert (Vgl. [Ben05], S. 104 f.)

nötigt werden. Das Langzeitgedächtnis unterstützt das Arbeitsgedächtnis indem es gewünschte Informationen liefert oder abspeichert.

2.1.4 Zusammenarbeit der Gedächtnisse

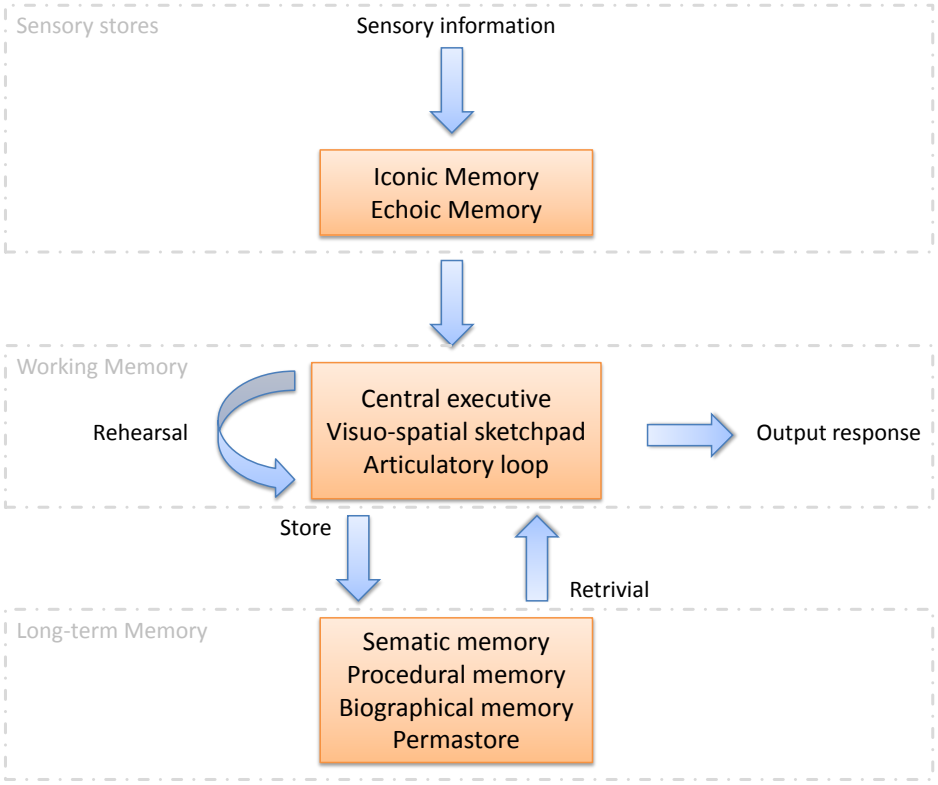

Abbildung 2.1: Schematisches Modell des „mutli-store memory“ nach Atkinson und Shiffrin (Vgl. [Ben05], S. 356)

Sensorische Informationen, werden im sensorischen Speicher für das Arbeitsgedächtnis bereitgehalten. Der Speicher ist in „iconic memory“, dem Bildspeicher, und dem „echoic memory“, dem auditiven Speicher, unterteilt. Im Arbeitsgedächtnis ist die zentrale Ausführungseinheit (*central executive*) für die Verarbeitung, Speicherung ins Langzeitgedächtnis (*store*), Aufarbeitung um Informationen nicht zu vergessen (*rehearsal*) und die Ausführung von Aktionen zuständig. Neben dem zentralen Master-System existieren zwei Sub-Systeme, eines für die Verarbeitung von auditiven (*articulatory loop*) und eines für die Verarbeitung von bildhaften Informationen (*visuo-spatial sketchpad*). Eine Besonderheit bei den Sub-Systemen ist, dass bildhafte Repräsentationen von Informationen parallel und auditive Repräsentationen nur sequenziell abgerufen werden können. Daraus lässt sich beispielsweise der Vorteil von Metapher erklären, da diese

bildhafte Repräsentationen sind, die parallel verarbeitet werden können. Vom Arbeitsgedächtnis können Informationen ins Langzeitgedächtnis geschrieben oder abgerufen werden. Im Langzeitgedächtnis wird in folgenden Speichern unterschieden:

- **Semantischer Speicher** (*semantic memory*)
  Speichert Informationen über die Bedeutung.
- **Prozeduraler Speicher** (*procedural memory*)
  Speichert Informationen über Handlungen wie Fahrrad fahren oder schreiben, also wie etwas gemacht wird.
- **Episodischer Speicher** (*episodic memory*)
  Hier werden Erfahrungen oder Informationen mit persönlichem Bezug gespeichert, wie etwa Geburtstage, der eigene Hochzeitstag oder auch die Beförderung zum Abteilungsleiter.
- **Permanentspeicher** (*permastore*)
  Hier werden Informationen gespeichert, die nicht mehr vergessen werden. Es sind elementare Informationen wie der eigene Name oder wo links und rechts ist.

2.2 Schemata

Sir Frederic Bartlett hat 1932 das Konzept der Schemata vorgestellt, um zu erklären, wie menschliche Wissensrepräsentation funktioniert und insbesondere deren Fehler zustande kommen. Die Probanden haben eine Geschichte gelesen, die sie nach einiger Zeit wiedergeben sollten. Das Besondere an der Geschichte war, dass sie den Erfahrungen der Probanden nicht ähnlich war. Beispielsweise eine Indianergeschichte bei der es um zwei Männer geht, die zusammen zur Seehundjagd gehen wollen. Das Ergebnis der Nacherzählung ergab, dass das Konzept der Geschichte behalten wurde, jedoch die bestimmten Details, besonders jene die kulturfremd waren, durch neue, den eigenen Erfahrungen besser angepasste Details, ersetzt wurden. So wurde aus einem Kanu ein Ruderboot oder aus Seehunde jagen fischen. Dadurch wurde klar, das Informationen in einem narrativen Kontext besser behalten werden als beispielsweise tabellarische Daten, da einzelne Details zwar verschwinden können, das Konzept aber erhalten bleibt. Aus diesem Grund wird in diesem Framework auf die Art der Kommunikation und der Art und Weise wie Artefakte, beispielsweise ein User Profile, erstellt werden, besonders viel Wert gelegt, da dies ganz elementar die Entwicklung beeinflusst.

2.3 Mentale Modelle

Mentale Modelle sind dynamische Repräsentationen von strukturellem oder funktionalem Wissen, die im Arbeitsgedächtnis, auf Basis von vorhandenem Wissen zum aktuellen Kontext aus dem Langzeitgedächtnis instanziiert werden. Da nach Johnson-Laird („mental models“, 1983) mentalen Modellen immer eine bildhafte Assoziation zugeordnet ist, können für bestimmte Kontexte adäquate mentale Modelle durch entsprechende Metaphern getriggert werden. Mentale Modelle sind sehr flexibel. Wenn kein konkretes Wissen über eine bestimmte Handlung vorhanden ist, wird versucht, ähnliche mentale Modell darauf anzuwenden und anzupassen. Mentale Modelle bestehen nicht aus festen Interaktionsabläufen, sondern aus dem Modell welches der Interaktion, aus Sicht des jeweiligen Betrachters, zugrunde liegt.
Um einen Benutzer optimal zu unterstützen, sollten Interfaces so gestaltet sein, dass sie den mentalen Modellen der Benutzer entsprechen. Mentale Modelle sind daher die Basis für das Interface-Design, da ohne Wissen über die mentalen Modelle der Benutzer kein Interface entwickelt werden kann, was den mentalen Modellen der Benutzer gerecht wird. Im Kontext der Entwicklung sollten die mentalen Modelle die das Team beispielsweise zu den Artefakten der Prozesse hat, identisch sein. Dadurch kann die Effizienz gesteigert und Verständnisprobleme gemindert werden. Man unterscheidet zwei Arten von mentalen Modellen:

- **Funktionale mentale Modelle**
  Es ist bekannt, wie man etwas benutzen muss, jedoch sind keine Informationen über Funktionsweise oder ähnlichem bekannt.

- **Strukturelle mentale Modelle**
  Es ist bekannt, wie etwas funktioniert (oder Glaube zu wissen), so können auch Fehler behoben oder unerwartete Ereignisse bewältigt werden.

2.4 Wahrnehmung

2.4.1 Visuelle Wahrnehmung (*feature integration theory*)

Die *feature integration theory*, kurz FIT, wurde von Ann Triesman in den 80'er Jahren entwickelt (siehe Abbildung 2.2. In der Theorie wird beschrieben, dass manche Prozesse bei der visuellen Wahrnehmung parallel und manche sequenziell ablaufen. So ist das Auflösen einer visuellen Szene in seine Bestandteile (präattentive Phase) wie Farbe, Position, Krümmung, Bewegung etc. ein paralleler Prozess. Die eigentliche bewusste Wahrnehmung ist Sequenziell (attentive Phase), dies bedeutet, dass der Aufmerksamkeitsfokus nur auf einen bestimmten Teil der Szene gerichtet werden kann. Die

Erkenntisse sind für die Gestaltung interaktive Systeme wichtig, da dadurch klar wird, dass Menschen Interfaces zwar als Ganzes wahrnehmen, aber nicht im Ganzen analysieren können. Dazu muss der Aufmerksamkeitsfokus über die aktuelle Szene „wandern“ und nach und nach analysieren. Wichtig ist an dieser Stelle, in welcher Reihenfolge der Benutzer vermutlich das Interface analysieren wird, um die wichtigtsten Interaktionspunkte frühzeitig zu erkennen.

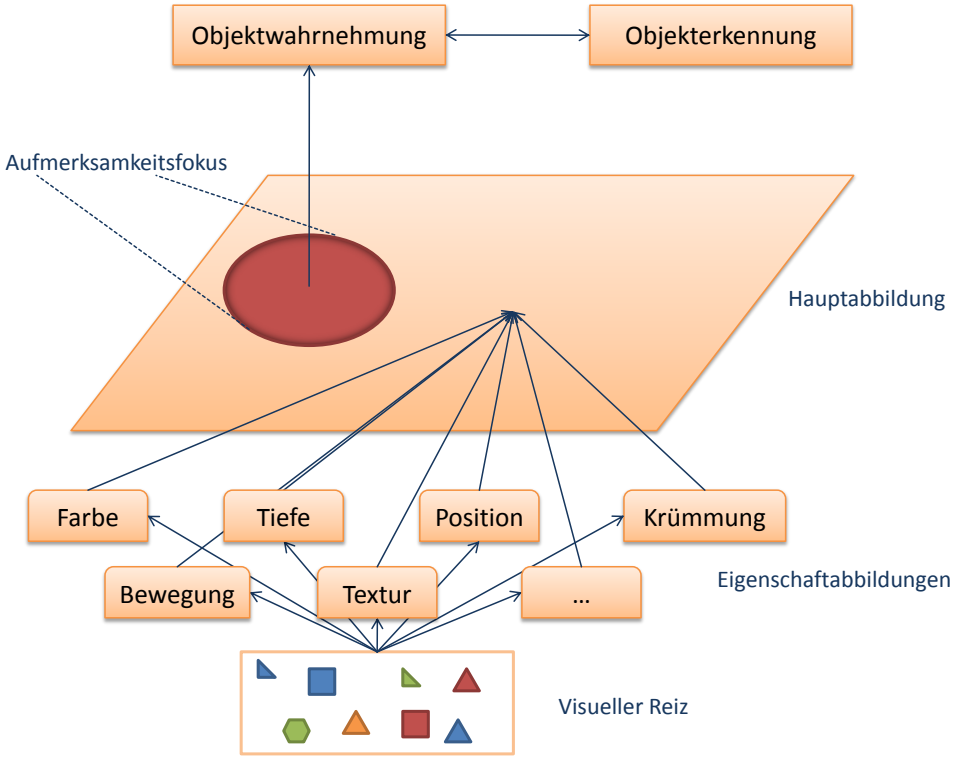

Abbildung 2.2: feature integration theory von Ann Treisman

2.4.2 Erkenntnisse aus der Gestaltpsychologie

Aus dem Fachbereich der Gestaltpsychologie und aus dem Bereich des Designs können einige Erkenntnisse verwendet werden, die als Grundregeln beim Entwurf neuer Prototypen verwendet werden können. Die Gestaltgesetze definieren klare Grenzen, in welchem Kontext etwas die Wahrnehmung beeinflusst. Diese Effekte können einerseits die Wahrnehmungsarbeit erhöhen, gut eingesetzt aber auch erheblich verbessern. Gestaltgesetze können den Aufmerksamkeitsfokus beeinflussen. So kann beispielsweise das Rastern des Interfaces für Benutzer und somit die Suche zur gewünschten Interaktion

beschleunigt werden, wenn das Interface klar unterteilt ist und die wichtigsten[2] Interaktionspunkte hervorgehoben sind. Einige Gestaltgesetze sind aus eigenen Erfahrungen her bekannt, vermutlich aber nicht bewusst.

- **Gesetz der Nähe**
  Elemente, die dicht beieinander stehen, werden als zusammengehörig aufgefasst.
- **Gesetz der Ähnlichkeit**
  Elemente, die sich ähnlich sind (Farbe, Größe, Form, etc.) werden als zusammengehörig aufgefasst.
- **Gesetz der guten Gestalt**
  Mehrere Elemente versucht das Gehirn zu einer Gestalt zu verbinden, wobei eine möglichst einfache und einprägsame bevorzugt wird.
- **Gesetz der guten Fortsetzung**
  Elemente, die auf einer durchgehenden Linie oder Kurve angeordnet sind, werden als zusammengehörig aufgefasst.
- **Gesetz der Geschlossenheit**
  Elemente, die von einer Linie umschlossen sind, werden als zusammengehörig aufgefasst.
- **Gesetz der Erfahrung**
  Elemente, die sich zu bekannten Gestalten zusammenfügen lassen, werden als solche erkannt, auch wenn diese unvollständig sind.
- **Gesetz des gemeinsamen Schicksals**
  Mehrere, sich gleichzeitig und gleichförmig bewegende Elemente, werden als zusammengehörig aufgefasst.

Neben den Gestaltgesetzen gibt es einige andere Faktoren, die Wahrnehmung beeinflussen, hierzu gehören Objekteigenschaften wie Größe, Farbe oder Position. Objekte werden als im Vordergrund stehend wahrgenommen, wenn sie

- eine warmen Farbton neben einem Objekt mit einem kalten Farbton stehen.
- ein anderes Objekt überlagern.
- größer sind als ein anderes Objekt.

[2] Wichtig meint in diesem Kontext, was am häufigsten benötigt wird oder (wie eine Not-Aus-Funktion im Atomkraftwerk) unabdingbar ist. Dies könnte beispielsweise die Hauptnavigation sein, diese sollte sich merklich von untergeordneten Elementen unterscheiden.

- der Farbton mehr Sättigung oder Kontrast besitzt als ein anderes Objekt.

Da in Europa von links oben nach rechts unten gelesen wird, erhält man an diesen Punkten die höchste Aufmerksamkeit. Einige Emotionen in Verbindung mit visuellen Reizen sind kulturabhängig. So werden Objekte am linken Bildschirmrand als kommend und am rechten Bildschirmrand als gehend wahrgenommen. Ein Linienverlauf von links unten nach rechts oben wird als aufsteigend, positiv, und umgekehrt als negativ wahrgenommen.

2.4.3 Goldener Schnitt

Der goldene Schnitt beschreibt ein Verhältnis, in dem das Kleine zum Großen, wie das Große zum Ganzen steht. Mathematisch gesehen ist dies ein Verhältnis von 1,618:1. Der goldene Schnitt kommt in der Natur häufig vor und diese Proportion wird vom Menschen als harmonisch und ästhetisch wahrgenommen. Dieser Effekt lässt sich meist sehr gut in Interfaces nutzen, indem die Inhalte und Objekte in diesem Verhältnis proportioniert werden.

3 Einführung in relevante Standards

3.1 ISO 9241

Die ISO 9241 ist ein internationaler Standard, der sich mit den „Ergonomischen Anforderungen für Bürotätigkeiten an Bildschirmgeräten“ [ISOd] beschäftigt. Die ISO 9241 definiert einige Begriffe aus dem Usability-Engineering, die hier verwendet werden, beispielsweise Gebrauchstauglichkeit. Der Standard ist in viele Teilbereiche aufgeteilt, so gibt es zum Beispiel Teile für die Anforderungen an Arbeitsplätze, Farbdarstellung, Anzeigegeräte, Tastaturen, Gebrauchstauglichkeit oder Grundsätze der Dialoggestaltung wie auch der Informationsdarstellung. Insgesamt umfasst der Standard siebzehn Teile.

Begriffsklärung

- **Gebrauchstauglichkeit**
  Die Gebrauchstauglichkeit definiert sich durch das Ausmaß, in dem ein bestimmtes Produkt durch bestimmte Benutzer in einem bestimmten Nutzungskontext genutzt werden kann, um bestimmte Ziele effektiv, effizient und zufriedenstellend erreichen.

- **Effektivität**
  Die Genauigkeit und Vollständigkeit mit der Benutzer ein bestimmtes Ziel erreichen.

- **Effizienz**
  Der im Verhältnis zur Genauigkeit und Vollständigkeit gesetzte Aufwand, mit dem Benutzer ein bestimmtes Ziel erreichen.

- **Zufriedenstellung**
  Freiheit von Beeinträchtigung und positive Einstellung gegenüber der Nutzung des Produktes.

- **Ziel**
  Ein angestrebtes Arbeitsergebnis.

- **Aufgabe**
  Die zur Zielerreichung erforderlichen Aktivitäten.

- **Arbeitssystem**
  Ein System, das aus Benutzern, Arbeitsmitteln, Arbeitsaufgaben und der physischen wie sozialen Umgebung besteht, um bestimmte Ziele zu erreichen.

3.2 ISO 13407

Die ISO 13407 (siehe Abbildung 3.1) ist ein internationaler Standard, der sich an Projektmanager richtet, die interaktive Systeme entwickeln (Vgl. [ISOa] S. 3). Der Fokus liegt auf der Planung und dem Management solcher Projekten, nicht etwa die Entwicklung direkt. Das Modell besteht, neben dem Eintritts- und Austrittspunkt, aus vier Phasen, welche inkrementell aufeinander aufbauen. Die ISO 13407 ist ein Vorgehensmodell für benutzerzentrierte Entwicklung. Aus diesem Grund wurde es für das Framework ausgewählt, um dieses in einem umfassenden Framework mit Softwareentwicklungsprozessesen zu vereinigen. Im folgenden Abschnitt werden die sechs Phasen nur kurz beschrieben, die einzelnen Sub-Ziele dieser Phase werden darauf folgend durch die ISO/PAS 18152 erläutert.

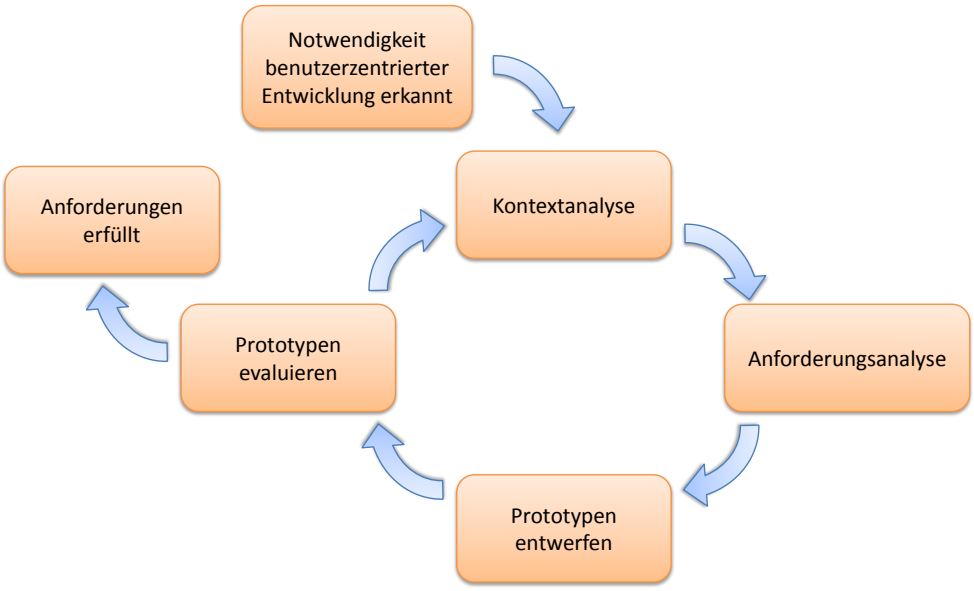

Abbildung 3.1: Vorgehensmodell der ISO 13407

- **Notwendigkeit benutzerzentrierter Entwicklung erkannt**
  Die Notwendigkeit benutzerzentrierter Entwicklung muss von allen beteiligten erkannt worden sein. Falls eine oder mehrere Personen nicht davon überzeugt sind, müssen diese entweder ausgetauscht oder überzeugt werden, anderenfalls sollte kein benutzerzentrierter Entwicklungsprozess durchgeführt werden.

- **Kontextanalyse**
  Im Rahmen der Kontextanalyse werden die Benutzer ihre Aufgaben, das Arbeitsumfeld und das Arbeitssystem analysiert. Der Umfang des Kontextes kann variieren, so wie die Relevanz mancher Faktoren. Beispielsweise muss bei einer Handy-Anwendung bedacht werden, dass Displays in der Sonne schlecht wahrzunehmen sind.

- **Anforderungsanalyse**
  Die Aufteilung zwischen Mensch und Computer, die Anforderungen der Benutzer, die organisatorischen Anforderungen und die zukünftige Benutzung des Systems wird beschrieben. Zusätzlich muss ein Modell der Benutzeraufgaben entwickelt werden. Die Wichtigkeit der Organisationsschnittstelle wird im IFIP (*International Fedoration of Information Processing*) Modell deutlich (siehe Abbildung 3.2). Dort gibt es zwei Instanzen der Organisationsschnittstelle, eine zur Verbindung

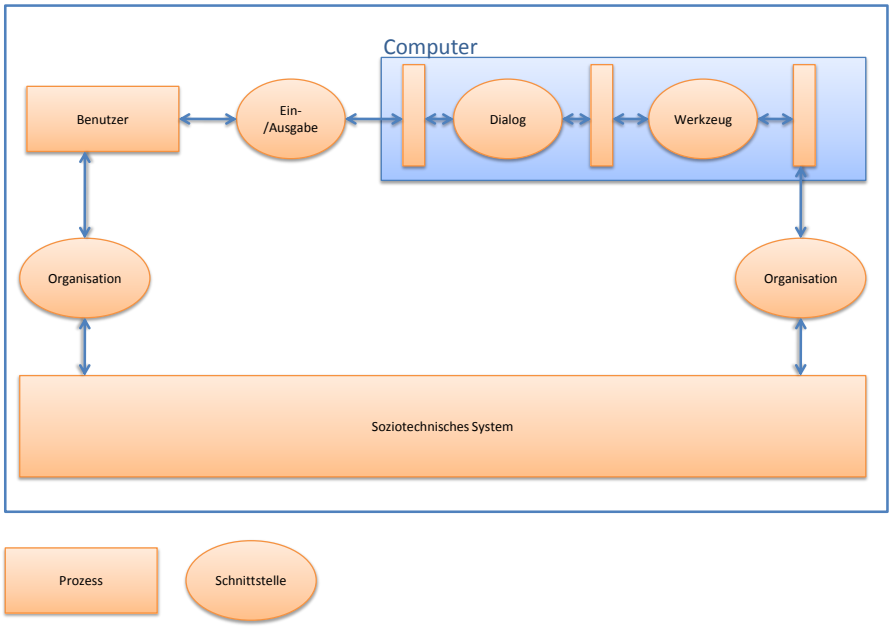

Abbildung 3.2: Das IFIP Modell besitzt zwei Organisationsschnittstellen

des Benutzers und eine zur Verbinung des Computers mit dem soziotechnischem System. Daher muss auch bei der Analyse darauf geachtet werden, in welchem Kontext diese beiden Einheiten stehen und was für die Kommunikation über diese Schnittstelle benötigt wird. Für den Benutzer kann da beispielsweise der Status in der Firma und für den Computer die Notwendigkeit eines Netzwerkkabels oder eine für das Unternehmennetzwerk valide IP-Addresse von belang sein.

- **Prototypen entwerfen**
  Auf Basis der Anforderungen sollen Prototypen entwickelt werden, hierbei sollen verschiedene Aspekte wie technische Einschränkungen, psychologische Grundlagen, Design Principles oder ähnliche relevanten Erkenntnisse berücksichtigt werden. Wichtig ist, dass nicht nur ein, sondern mehrere Prototypen entwickelt werden.

- **Prototypen evaluieren**
  Die Evaluation soll zusammen mit den Benutzern oder deren Stellvertretern durchgeführt werden und feststellen, ob die definierten Anforderungen erfüllt sind. Ein sehr wichtiger Punkt ist, wenn die Anforderungen nicht erfüllt werden, der Prozess bei der Analyse des Nutzungskontextes neu beginnt und nicht etwa bei der Prototyping-Phase. Die ist Notwendig, da sich die Anforderungen durch die Nutzungskontext- oder Anforderungsanalyse während der Entwicklung geändert haben können oder nicht richtig erfasst wurden.

- **Anforderungen erfüllt**
  Alle definierten Anforderungen der Benutzer wie der Organisation an die Gestaltungslösung sind erfüllt und beachten die gewonnenen Erkenntnise der Kontext- und Anforderungsanalyse.

3.3 ISO/PAS 18152

Die ISO/PAS 18152 [ISOe] definiert in Abschnitt 3 Basispraktiken, die als Mindesanforderung benötigt werden, um gebrauchstaugliche Systeme zu entwickeln. Die Basispraktikten sind Kategorien zugeordnet, die dem Vorgehensmodell der ISO 13407 entsprechen. PAS (publicly available specification) bedeutet, dass diese ISO noch nicht zum Standard erhoben, aber schon mit einer 50% Mehrheit abgestimmt wurde. Im Dezember 2009 wird abgestimmt, ob die 18152 zum Standard erhoben wird. Die geforderten Basispraktiken sind nur eine Zieldefinition, in der ISO/PAS 18152 wird nicht definiert, wie diese erreicht werden können oder wie festgestellt werden kann, ob diese Ziele vollständig erreicht worden sind. Die geforderten Basispraktiken der ISO/PAS

18152 werden im Framezork zur Qualiätssicherung auf Seite des Usability Engineerings verwendet. Im Framwork werden Methoden und Techniken vorgestellt, mit denen die definierten Ziele erreicht werden können. Ein wichtiger Aspekt ist auch die Evaluierung der Prozesse, da festgestellt werden muss, ob die Ziele erreicht wurden. Hierzu werden später auch Hilfestellungen gegeben.

Ziele der Hauptprozesse

Kontextanalyse (context of use)

Die Kontextanalyse zielt darauf ab, die Arbeitsumgebung und die Aufgaben des Benutzers zu analysieren. Dazu gehören die physikalische, psychische und organisatorische Umgebung des Benutzers, so wie die Aufgaben, die ihm in der Unternehmung zugeteilt sind. Nicht außer Acht zu lassen sind auch Eigenschaften, wie die kulturelle oder ethnische Herkunft, die unter Umständen zu Kommunikations- oder Verständnisproblemen führen können (z.B. bei Metaphern). Als Ergebnis der Kontextanalyse sollte klar sein, welche Aufgaben die Benutzer haben, welche Interaktion es mit anderen Benutzer und dem System gibt und durch was die Benutzer beeinflusst werden.

Basispraktiken

- Der Rahmen für den Nutzungskontext wird abgesteckt.
- Aufgaben und Arbeitssystem werden analysiert.
- Die Eigenschaften der Benutzer werden beschrieben.
- Die kulturelle Umgebung, die Organisationsstruktur sowie Managementmodelle werden beschrieben.
- Eigenschaften von Geräten oder Systemen, die mit dem zukünftigen System gemeinsam genutzt werden sollen, werden beschrieben.
- Der Ort, die vorhandene Arbeitsplatzaustattung sowie Umgebungsfaktoren werden beschrieben.
- Die Auswirkungen des Nutzungskontextes werden untersucht/abgeschätzt.

Anforderungsanalyse (user requirements)

Ziel der Anforderungsanalyse ist es, die Anforderungen der Benutzer an das System festzustellen. Es sollten die Bedürfnisse aller Benutzergruppen berücksichtigt werden.

Basispraktiken

- Es erfolgt eine Funktionsaufteilung zwischen Mensch, Maschine und Organisation, basierend auf der bestmöglichen Eignung für die gegebene Funktion.
- Ein praktisches Modell der Benutzeraufgaben wird entwickelt, basierend auf den Anforderungen, dem Nutzungskontext, der Funktionsteilung sowie anderen Rahmenbedingungen.
- Für die benutzerrelevanten Anteile des Systems werden Lösungen (Konzepte, Designs) produziert, die die Anforderungen, den Nutzungskontext sowie das ergonomische Wissen berücksichtigen.
- Die zukünftige Benutzung des Systems wird beschrieben.
- Das Design sowie Sicherheitsaspekte werden basierend auf Evaluationsfeedback weiterentwickelt.

Entwurfsprozess (produce design solutions)

Im Entwurfsprozess werden die Daten der Benutzer berücksichtigt, um einen ergonomischen Prototypen zu entwickeln. Es werden mehrere Prototypen und Paradigmen diskutiert und zusammen mit den Benutzern evaluiert.

Basispraktiken

- Das erwartete Systemverhalten in Bezug auf die Benutzerinteraktion wird spezifiziert und vereinbart.
- Die Benutzeranforderungen an das System werden explizit formuliert.
- Die Benutzeranforderungen werden (z.B. im Hinblick auf Verbesserungsmöglichkeiten) analysiert.
- Messbare Kriterien zur Bewertung des Systems im Nutzungskontext werden definiert und vereinbart.
- Die Anforderungen werden Projekt-Stakeholdern zur Weiterverwendung in der Entwicklung und dem Betrieb des Systems weitergegeben.

Evaluationsprozess (evaluation of use)

Im Evaluationsprozess werden Informationen über die Benutzung des Systems gesammelt. Dazu gehören Beobachtungen der User am System, Evaluierung hinsichtlich der Anforderungen und Intensivtests durch Key-User.

Basispraktiken

- Die Evaluation wird geplant.
- Die Rahmenbedingungen der Evaluation werden identifiziert und betrachtet.
- Es wird sichergestellt, dass das System reif für eine Evaluation ist.
- Die Evaluation und Auswertung werden entsprechend dem Evaluationsplan durchgeführt.
- Frühe Prototypen werden evaluiert, um die Benutzeranforderungen an das System herauszufinden.
- Prototypen werden evaluiert, um das Konzept und Design zu optimieren.
- Das System wird evaluiert, um zu überprüfen, ob alle stakeholder- und organisatorischen Anforderungen erfüllt sind.
- Das System wird evaluiert um sicherzustellen, dass das erforderliche Vorgehen angewandt wurde.
- Das System wird in der realen Nutzungsumgebung evaluiert um sicherzustellen, dass es dort auch weiterhin die Anforderungen der Benutzer sowie der Organisation erfüllt.

3.4 ISO 12207

3.4.1 Begriffsdefinition

Hier werden Begriffe definiert, wie sie in der ISO verwendet werden, diese Definitionen gelten nur für die ISO 12207 und nicht für den Rest dieser Arbeit.

- **Benutzer**
  Ein Individuum oder eine Organisation, die eine bestimme Funktionalität des operativen Systems benutzt.
- **Entwickler**
  Eine Organisation, die Entwicklungsprozesse durchführt. Im Entwicklungsprozess sollten Anforderungsanalyse, Design und Akzeptanztests vorhanden sein.
- **Software-Unit**
  Ein Software-Unit ist die kleinste Software-Einheit, ein kleines Stück kompilierbarer Code.

- **Software-Item**
  Ein Software-Item besteht aus mehreren Software-Units und ist daher eine niedrigere Granularitätsstufe.

- **Software-Komponente**
  Eine Softwarekomponente wird in der Regel im Grobentwurf spezifiziert und beschreibt unter anderem Schnittstellen.

Die ISO 12207 [ISOb] beschäftigt sich mit dem Lebenszyklus von Software (*Software lifecycle processes*). Der Zyklus beschreibt die Beschaffung neuer Software bis hin zur Wartung nach Inbetriebnahme. Es werden Primärprozesse, unterstützende und organisatorische Prozesse definiert. Der Vollständigkeit halber werden erst alle Subprozesse erwähnt, wobei später nur auf die relevanten Prozesse für Entwicklung und Planung eingegangen wird. Die ISO 12207 soll die Qualität des Frameworks im Bereich der Softwareentwicklung sichern. So sollen die entwickelten Systeme korrekt, robust, effizient, wieder verwendbar, wartbar und portabel sein und nicht ausschließlich der Anforderungen an Gebrauchstauglichkeit genügen.

3.4.2 Primärprozesse

- **Beschaffungsprozess**
  Beschaffung neuer Software

- **Lieferungsprozess**
  Lieferung neuer Software aus Sicht des Lieferanten

- **Entwicklungsprozess**
  Entwicklung neuer Software

- **Betriebsprozess**
  Inbetriebnahme und aktives Arbeiten mit neuer Software

- **Wartungsprozess**
  Wartung von Software

3.4.3 Unterstützende Prozesse

- **Dokumentationsprozess**
  Definiert die Dokumentation einzelner Prozesse und Aktivitäten.

- **Konfigurations-Managementprozess**
  Definiert die Aktivitäten des Konfigurationsmanagements.

- **Qualitätssicherheitsprozess**
  Definiert Aktivitäten, um die Softwareprodukte und Prozesse hinsichtlich der Anforderungen zu prüfen.

- **Verifikationsprozess**
  Definiert Aktivitäten zur Verifizierung von Softwareprodukten aus Sicht des Beschaffers, des Lieferanten und unabhängigen Dritten.

- **Validierungsprozess**
  Definiert Aktivitäten zur Validierung von Softwareprodukten aus Sicht des Beschaffers, des Lieferanten und unabhängigen Dritten.

- **Joint-review Prozess**
  Definiert Aktivitäten zur Evaluation des Status und des Produktes einer Aktivität.

- **Audit-Prozess**
  Definiert Aktivitäten zur Einhaltung der Anforderungen, Modelle und Verträge.

- **Problembehebungsprozess**
  Definiert einen Prozess zur Extraktion, Analyse und Beseitigung von Problemen.

3.4.4 Organisatorische Prozesse

- **Managementprozess**
  Definiert grundlegende Aktivitäten des Managements, eingeschlossen die Aktivitäten des Projektmanagements bezüglich der Ausführung des Softwareentwicklungsprozesses.

- **Infrastrukturprozess**
  Definiert grundlegende Aktivitäten zur Etablierung eines Entwicklungszyklus, der einer bestimmten Struktur entspricht.

- **Optimierungsprozess**
  Definiert grundlegende Aktivitäten, die zur Optimierung der einzelnen Prozesse in einem Unternehmen benötigt werden.

- **Schulungsprozess**
  Definiert Aktivitäten, die für adäquate Schulungsmaßnahmen benötigt werden.

Die einzelnen Subprozesse beinhalten wiederum Prozesse, die aus einem oder mehreren Tasks bestehen. Aufgrund der Abgrenzung dieser Arbeit werden nur die Prozesse und Tasks der Entwicklung detailliert.

3.4.5 Entwicklungsprozess

Der Softwareentwicklungsprozess enthält die Aktivitäten und Aufgaben der Benutzer. Es werden Aktivitäten zur Prozessimplementation, System- und Softwareanforderungen, System- und Softwarearchitektur sowie Programmierung, Tests und Installation definiert.

Aktivitäten des Entwicklungsprozess

- **Prozessimplementation**
  - Der Entwickler soll Standards, Methoden, Techniken und Programmiersprachen (sofern nicht vertraglich vorgegeben) auswählen und anpassen, die innerhalb des Unternehmens dokumentiert und etabliert sind.
  - Der Entwickler soll Ausführungspläne für den Entwicklungsprozess entwickeln, die die nötigen Standards, Methoden, Tools und Aktivitäten beinhalten und die Verantwortlichkeiten verteilen.
- **Anforderungen an das System ermitteln**
  - Die genauen Ziele des zu entwickelnden Systems sollen analysiert werden, um die Anforderungen an das System zu ermitteln. Die Systemanforderungen sollen Funktionen, Fähigkeiten, Business-, organisatorische sowie Benutzeranforderungen, Betriebssicherheit, Zugriffssicherheit, Ergonomie, Schnittstellen-, operative und Wartungsanforderungen wie auch Designabgrenzungen und Qualifizierungsanforderungen enthalten. Die Systemanforderungen sollen dokumentiert werden.
  - Die Systemanforderungen sollen nach den folgenden Kriterien evaluiert und dokumentiert werden: Nachvollziehbarkeit der Akquisitionsanforderungen, Übereinstimmung mit den Akquisitionsanforderungen, Testbarkeit, Durchführbarkeit des systemarchitektuellem Design, Durchführbarkeit von Betrieb und Wartung.
- **Design für die Systemarchitektur entwickeln**
  - Eine top-level Architektur des Systems soll etabliert werden. Die Architektur soll Einzelheiten der Hardware, Software und manuelle Bedienungen enthalten. Es soll sichergestellt sein, dass alle Systemanforderungen erfüllt werden. Die Systemarchitektur und die Systemanforderungen sollen dokumentiert werden.

  - Die Systemarchitektur soll nach den folgenden Kriterien evaluiert und dokumentiert werden: Nachvollziehbarkeit der Systemanforderungen, Übereinstimmung mit den Systemanforderungen, Angemessenheit der angewandten Design-Standards und Methoden, Realisierbarkeit der Software-Items, Durchführbarkeit von Betrieb und Wartung.

- **Anforderungen an die Software ermitteln**
  Für jedes Software- oder Softwarekonfigurations-Item (sofern identifiziert) sollen folgende Tasks durchgeführt werden:
  - Der Entwickler soll Softwareanforderungen erarbeiten und dokumentieren, die der folgenden Liste von Qualitätsmerkmalen entsprechen:
    * funktionale Anforderungen und Fähigkeiten, die Leistung, physikalische Eigenschaften und Umgebungsfaktoren berücksichtigen, in der das Softwareelement ausgeführt wird
    * interne und externe Schnittstellen zum Software-Item
    * Qualifizierungsanforderungen
    * Sicherheitsanforderungen, inklusive derer, die einen Bezug zu Methoden des Betriebs, Wartung und Umgebungseinflüsse besitzen
    * Sicherheitsanforderungen die die Zugriffssicherheit betreffen, ebenso diese, die solche beeinflussen.
    * ergonomische Anforderungen der Benutzer, inklusive der „von Hand“ durchgeführten Prozesse, Interaktionen mit Equipment, Einschränkungen der Benutzer, Felder, die besondere Aufmerksamkeit und Konzentration erfordern und so sensitiv gegenüber menschlichen Fehlern sind
    * Datendefinition und Datenbankanforderungen
    * Installations- und Akzeptanzanforderungen an die auszuliefernde Software auf Produktiv- und Wartungsseite
    * Benutzerdokumentation
    * Benutzer-Operations und -Ausführungs Anforderungen
    * Benutzer-Wartungs-Anforderungen
  - Die Systemarchitektur soll nach den folgenden Kriterien evaluiert und dokumentiert werden: Nachvollziehbarkeit der Systemanforderungen und Systemdesign, externe Konsistenz mit den Systemanforderungen, interne Konsistenz, Testbarkeit, Realisierbarkeit des Softwaredesigns, Durchführbarkeit von Betrieb und Wartung

  - Der Entwickler soll Joint-Reviews durchführen

- **Architektur-Design für die Software entwickeln**
  Für jedes Software- oder Softwarekonfigurations-Item (sofern identifiziert) sollen folgende Tasks durchgeführt werden:
  - Der Entwickler soll die Anforderungen für die Softwareelemente in eine top-level Architektur überführen, die die einzelnen Softwarekomponenten beschreibt. Es soll sichergestellt sein, das jede Anforderung in den Softwarekomponenten berücksichtigt wird. Die Architektur soll dokumentiert werden.
  - Der Entwickler soll ein top-level Design der externen Schnittstellen so wie der internen Kommunikation zwischen den Softwarekomponenten entwickeln und dokumentieren.
  - Der Entwickler soll ein top-level Design der Datenbank entwickeln und dokumentieren.
  - Der Entwickler soll erste Vorabversionen der Benutzerdokumentation entwickeln und dokumentieren.
  - Der Entwickler soll vorab einige Evaluationsanforderungen definieren und dokumentieren und für die Softwareintegration planen.
  - Der Entwickler soll die Architektur der Softwarekomponenten, das Datenbankdesign und die Schnittstellen nach folgenden Kriterien analysieren:
    * Nachvollziehbarkeit der Anforderungen an eine Softwarekomponente
    * Externe Konsistenz mit den Anforderungen an die Softwarekomponente
    * Interne Konsistenz zwischen den Softwarekomponenten
    * Angemessene Verwendung der Design-Methoden und Standards
    * Durchführbarkeit des detaillierten Designs
    * Durchführbarkeit von Betrieb und Wartung

    Der Entwickler soll die Ergebnisse dokumentieren.
  - Der Entwickler soll Joint-Reviews durchführen.

- **Detailliertes Design für die Software entwickeln**
  Für jedes Software- oder Softwarekonfigurations-Item (sofern identifiziert) sollen folgende Tasks durchgeführt werden:
  - Der Entwickler soll für jede Softwarekomponente ein detailliertes Design durchführen. Die Softwarekomponenten sollen so verfeinert werden, dass sie

programmiert, kompiliert und evaluiert werden können. Es soll sichergestellt werden, dass alle Softwareanforderungen erfüllt sind. Das detaillierte Design soll dokumentiert werden.

- Der Entwickler soll ein detailliertes Design für die Schnittstellen nach aussen und zwischen den Softwarekomponenten entwickeln und dokumentieren. Das detaillierte Design soll es ermöglichen, dass Schnittstellen ohne weitere Informationen programmiert werden können.
- Der Entwickler soll ein detailliertes Design für die Datenbank entwickeln und dokumentieren.
- Der Entwickler soll, sofern nötig, die Benutzerdokumentation überarbeiten.
- Der Entwickler soll Anforderungen an die Evaluation der Softwarekomponenten entwickeln und dokumentieren so wie einen Plan für einen Softwaretest entwickeln. Die Anforderungen der Evaluation soll die Softwareanforderungen bis an die Grenzen belasten.
- Der Entwickler soll die Evaluationsanforderungen und den Plan für die Softwareintegration überarbeiten.
- Der Entwickler soll das detaillierte Design und die Testanforderungen hinsichtlich der folgenden Kriterien evaluieren:
  - Nachvollziehbarkeit der Anforderungen an die Software-Items
  - Externe Konsistenz zum architektuellen Design
  - Interne Konsistenz zwischen den Softwarekomponenten und Software-Units
  - Angemessene Verwendung der Design-Methoden und Standards
  - Durchführbarkeit der Tests
  - Durchführbarkeit von Betrieb und Wartung

  Der Entwickler soll die Ergebnisse dokumentieren.
- Der Entwickler soll Joint-Reviews durchführen.

- **Softwareprogrammierung und -testing**
  Für jedes Software- oder Softwarekonfigurations-Item (sofern identifiziert) sollen folgende Tasks durchgeführt werden:
  - Der Entwickler soll jede Software-Unit und jede Datenbank entwickeln und dokumentieren sowie für diese Testfälle Daten bereit stellen.

  - Der Entwickler soll jede Software-Unit und jede Datenbank testen, um sicherzustellen dass die Anforderungen erfüllt werden. Das Ergebnis soll dokumentiert werden.
  - Der Entwickler soll, sofern nötig, die Benutzerdokumentation überarbeiten.
  - Der Entwickler soll die Testanforderungen vervollständigen und die Softwareintegration planen.
  - Der Entwickler soll den Software-Code evaluieren. Die Ergebnisse sollen den folgen Kriterien entsprechen:
    * Nachvollziehbarkeit der Anforderungen und dem Design des Software-Items.
    * Externe Konsistenz mit den Anforderungen und dem Design des Software-Items.
    * Interne Konsistenz zwischen den Anforderungen der Software-Units.
    * Testabdeckung der Software-Units.
    * Verhältnismäßigkeit der Coding-Methoden und angewandten Standards.
    * Durchführbarkeit der Softwareintegration und Tests.
    * Durchführbarkeit von Betrieb und Wartung.

    Die Ergebnisse sollen dokumentiert werden.

- **Softwareintegration**
  Für jedes Software- oder Softwarekonfigurations-Item (sofern identifiziert) sollen folgende Tasks durchgeführt werden:
  - Der Entwickler soll ein Integrationsplan entwickeln, um die Software-Units und Softwarekomponenten in die Software-Items zu integrieren.
  - Der Entwickler soll die Software-Units und Softwarekomponenten integrieren und testen, ob diese Aggregation mit dem Integrationsplan übereinstimmt. Es soll sichergestellt werden, dass jede Aggregation konform zu den Anforderungen des Software-Items ist. Die Integration und die Tests sollen dokumentiert werden.
  - Der Entwickler soll, sofern nötig, die Benutzerdokumentation überarbeiten.
  - Der Entwickler soll Integrationsplan, Design, Code, Tests, Testergebnisse und Benutzerdokumentation nach den folgenden Kriterien evaluieren:
    * Nachvollziehbarkeit der Systemanforderungen
    * Externe Konsistenz mit den Systemanforderungen

        * Interne Konsistenz
        * Testabdeckung der Anforderungen des Software-Items
        * Verhältnismäßigkeit der verwendeten Teststandards und Methoden
        * Übereinstimmung mit den erwarteten Ergebnissen
        * Durchführbarkeit der Softwarequalifikationstests
        * Durchführbarkeit von Betrieb und Wartung

      Die Ergebnisse der Evaluation sollen dokumentiert werden.
    - Der Entwickler soll Joint-Reviews durchführen.

- **Softwarequalitätstest**
  Für jedes Software- oder Softwarekonfigurations-Item (sofern identifiziert) sollen folgende Tasks durchgeführt werden:
    - Der Entwickler soll Qualifikationstests, in Übereinstimmung mit den Softwareanforderungen der Software-Items, durchführen.
    - Der Entwickler soll, sofern nötig, die Benutzerdokumentation überarbeiten.
    - Der Entwickler soll Design, Code, Tests, Testergebnisse und Benutzerdokumentation nach den folgenden Kriterien evaluieren:
        * Testabdeckung der Anforderungen des Software-Items
        * Übereinstimmungen mit den erwarteten Ergebnissen
        * Durchführbarkeit von Systemintegration und Test, wenn durchgeführt
        * Durchführbarkeit von Betrieb und Wartung

      Die Ergebnisse der Evaluation sollen dokumentiert werden.
    - Der Entwickler soll Audits unterstützen und die Ergebnisse dieser dokumentieren. Wenn Hardware und Software noch in der Entwicklung sind, sollen die Audits bis zum Softwarequalitätstest verschoben werden.
    - Folgend auf erfolgreiche Durchführung der Audits, sofern durchgeführt, soll der Entwickler die auslieferbare Software für Systemintegration, Systemqualifikationstests, Softwareinstallation oder Software Akzeptanzunterstützung, sofern möglich, vorbereiten.

- **Systemintegration**
    - Die Softwarekonfigurations-Items, inklusive der Hardwarekonfigurations-Items und anderer Systeme, sofern nötig, sollen integriert werden. Die aggregierten Einheiten sollen gegen ihre Anforderungen getestet werden:

  - Für jede Qualitätsanforderung an das System, soll eine Reihe Tests, Testfälle und Testprozeduren entwickelt und dokumentiert werden. Der Entwickler soll sicherstellen, dass das System bereit für die Systemqualifizierung ist.
  - Das integrierte System soll nach den folgenden Kriterien evaluiert werden:
    * Testabdeckung der Systemanforderungen
    * Verhältnismäßigkeit der Testmethoden und verwendeten Standards
    * Übereinstimmung mit den erwarteten Ergebnissen
    * Durchführbarkeit der Systemqualifikationstests
    * Durchführbarkeit von Betrieb und Wartung

    Die Ergebnisse sollen dokumentiert werden.

- **System hinsichtlich der Anforderungen testen**
  - Die Systemqualifikationstests sollen in Übereinstimmung mit den spezifizierten Systemanforderungen durchgeführt werden. Es soll sichergestellt werden, dass jede Implementation der Systemanforderungen erfüllt und bereit für die Auslieferung ist. Die Qualifikationstests sollen dokumentiert werden.
  - Das System soll nach folgenden Kriterien evaluiert werden:
    * Testabdeckung der Systemanforderungen
    * Übereinstimmung mit den erwarteten Ergebnissen
    * Durchführbarkeit von Betrieb und Wartung

    Die Ergebnisse sollen dokumentiert werden.
  - Der Entwickler soll Audits durchführen und die Ergebnisse dokumentieren.
  - Folgend auf erfolgreiche Durchführung der Audits, sofern durchgeführt, soll der Entwickler die auslieferbare Software überarbeiten und für die Installation und Akzeptanzunterstützung vorbereiten. Zusätzlich soll eine Grundlinie für das Design und den Code für jedes Softwarekonfigurations-Item geschaffen werden.

- **Softwareinstallation**
  - Der Entwickler soll einen Plan zur Installation der Software in der Zielumgebung entwickeln, wie sie vertraglich definiert ist. Die Informationen und Ressourcen, die für die Installation benötigt werden, sollten verfügbar sein. Der Entwickler soll den Kunden im Rahmen der vertraglichen Vereinbarung bei Installation unterstützen. Wenn alte Systeme ersetzt werden oder ein

paralleler Betrieb stattfindet, soll der Entwickler den Kunden im vertraglich vereinbarten Rahmen unterstützen. Die Installationsplan soll dokumentiert werden.

- Der Entwickler soll die Software gemäß dem Installationsplan installieren. Es soll sichergestellt werden, das der Software-Code, die Datenbanken gemäß der Spezifikation im Vertrag initialisiert, ausgeführt und terminiert werden. Der Installationsprozesse und Ergebnisse sollen dokumentiert werden.

- **Unterstützung bei der Softwareakzeptanz**
  - Der Entwickler soll die Akzeptanz-Reviews und Tests des Kunden unterstützen. Die Akzeptanz-Reviews sollen die Ergebnisse der Joint-Reviews, Audits, Softwarequalifikationstests und Systemqualifikationstests (sofern durchgeführt) berücksichtigen. Die Ergebnisse des Reviews und der Tests sollen dokumentiert werden.
  - Der Entwickler soll die Softwareentwicklung beenden und wie im Vertrag vereinbart ausliefern.
  - Der Entwickler soll einführende und weiterführende Schulungen soweit unterstützen, wie es vertraglich vereinbart ist.

4 Integrations-Möglichkeiten der Standards

Die Standards aus den Bereichen der Softwareentwicklung und dem Usability Engineering weisen verschiedene Perspektiven auf, so dass sich diese gegenseitig ergänzen können, um die Qualität des Entwicklungsprozesses zu erhöhen. Die ISO 12207 als Standard der Softwareentwicklung hat den Fokus auf funktionalen Anforderungen, die das technische Subsystem und die Software betreffen. Benutzer, Nutzungskontext und die Organisationsstruktur, aus denen sich hauptsächlich non-funktionale Anforderungen ableiten lassen, werden dort nicht oder nur unzureichend berücksichtigt. Die ISO 13407 und die ISO/PAS 18152 hingegen fokussieren Benutzer, Nutzungskontext und Organisationsstruktur, jedoch nicht das technische Subsystem. Ebenso existiert in der ISO 12207 keine konkrete Prototyping Phase, in der verschiedene Prototypen entwickelt und diskutiert werden, sondern es wird „nur“ von Software- und Systemdesign gesprochen. Hingehen im Usability Engineering ist die Prototyping Phase ein wichtiger Bestandteil, in der mehrere Gestaltungslösungen entwickelt und durch Benutzer evaluliert werden. Qualitätsansprüche aus der Softwareentwicklung, wie etwa Skalierbarkeit, Wartbarkeit etc., unterscheiden sich deutlich von denen des Usability Engineerings, wie etwa Aufgabenangemessenheit, Erwartungskonformität oder Fehlertoleranz.
Karsten Nebe hat in seiner Dissertation über die Integration von Usability Engineering und Software Engineering eine Tabelle erarbeitet, die die einzelnen Aktivitäten der ISO 12207 im Softwareentwicklungsprozess und die Aktivitäten vom Vorgehensmodell der 13407 gegenüberstellt. Er räumt ein, dass keine festen Integrationspunkte existieren, diese also verschieden interpretiert werden können. Nebe kommt zu dem Ergebnis, dass trotz Fehlen fester Integrationspunkte, eine Integration stattfinden kann.

ISO/IEC 12207 Sub-Process: Development	Common Activities	DIN EN ISO 13407
Requirements Elicitation	Requirement Analysis	Context of Use User Requirements
System Requirements Analysis Software Requirements Analysis	Software Specification	Produce Design Solutions

System Architecture Design Software Design Software Construction Software Integration	Software Design and Implementation	n/a
Software Testing System Integration	Software Validation	Evaluation of Use
System Testing Software Installation	Evaluation	Evaluation of Use

Tabelle 4.1: Integrationspunkte über identische Aktivitäten der Softwareentwicklung (ISO 12207) und des Usability Engineerings (ISO 13407) (Vgl. [Kar08], S.59)

Eine Integration kann nicht nur über Prozesse, sondern auch über gemeinsame Artefakte, Methoden und Techniken stattfinden. Wenn eine Integration über Prozesse gewählt wird, ist die Abstraktion relativ groß, so dass die einzelnen Aktivitäten und Artefakte dem aktuellen Erkenntnisstand angepasst werden können. Eine Integration über Artefakte ist dann möglich, wenn diese nicht als ein konkretes Artefakt, sondern als Projekt-Dokomente, wie etwa System-, Software- oder Benutzerdokumentation angesehen werden. Solche Projekt-Dokumente sind in Usability- und Softwareentwicklungsprozessen etabliert. Durch die Vorgabe welche Dokumentationen existieren sollen, bleibt freigestellt, wie die konkrete Ausprägung dieser ist. Aus diesem Grund wird in diesem Framework eine Integration über gemeinsame Prozesse und Projekt-Dokumente gewählt. Die Prozesse der Standards und die Projekt-Dokumente werden in diesem Abstraktionsgrad voraussichtlich langlebiger sein als einzelne Methoden oder Techniken, die verhältnismäßig schnell nicht mehr „state of the art“ sein können. Als Ausgangspunkt der Integration wird das Vorgehensmodell der ISO 13407 gewählt, da sich darin alle Aktivitäten der ISO/PAS 18152 und die meisten Aktivitäten der ISO 12207 in die Prozesskategorien integrieren lassen. Die Aktivitäten der ISO 12207 unterliegen keiner vorgegeben Reihenfolge, dadurch wird der Standard bei der Integration nicht verletzt. Trotz gemeinsamer Aktivitäten in der Anforderungsermittlung (siehe Abbildung 4.1) in der ISO 12207 und der ISO 13407, existiert ist der ISO 12207 kein Prozess, der die organisatorischen Anforderungen (siehe Abbildung 3.2) ermittelt oder dokumentiert. Für die weiteren Prozesse der ISO 13407 („Kontextanalyse“, „Anforderungsanalyse (Benutzer)“, „Gestaltungslösungen entwickeln“, „Gestaltungslösungen evaluieren“) existieren mindestens zwei Prozesse der ISO 12207, die sich darin integrieren lassen. Da sich die 13407 nicht konkret mit dem Prozess der Entwicklung von Software beschäftigt, lassen sich die Prozesse der ISO 12207, die sich mit der Entwicklung beschäftigen, einzig im

Prozess „Gestaltungslösungen evaluieren“ integrieren. Für ein erweitertes Vorgehensmodell wäre es möglich, diese in einem eigenen Prozess mit zusätzlicher Evaluation der entwickelten Komponenten zu verschieben, um eine klare Trennung zwischen der Entwicklung und Evaluation von Gestaltungslösungen und tatsächlichen Komponenten zu schaffen.

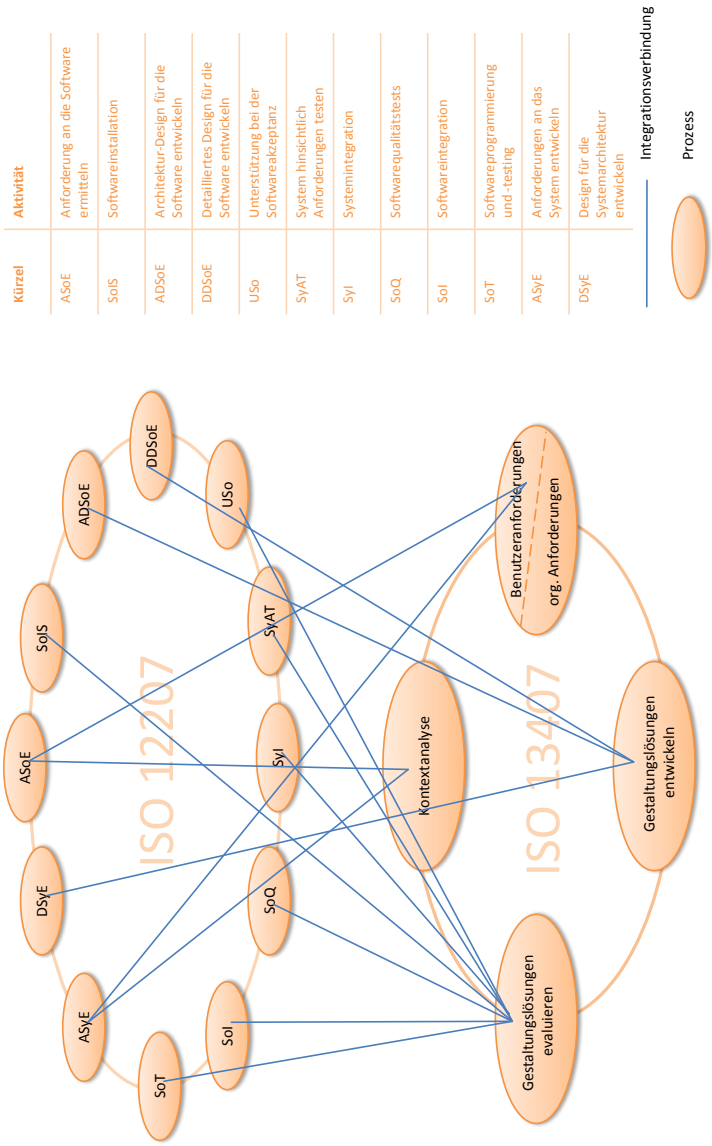

Abbildung 4.1: Mögliche Durchstoßpunkte zwischen der Softwareentwicklung und dem Usability Engineering

5 Scrum

Scrum ist ein agiles Vorgehensmodell zur Entwicklung von Software, welches sich in einigen Unternehmen etabliert hat. Scrum kann verschieden aufgefasst werden: Einerseits können die Regeln strikt befolgt und Scrum so als ein Ziel der Entwicklung gesehen werden, was das Mischen mit anderen Verfahren schwieriger gestaltet. Die andere und hier bevorzugte Möglichkeit ist, Scrum als Methode zu sehen, die durch verschiedene, je nach Situation durchgeführte, Techniken ihre Ziele erreicht. Dies ermöglicht es, sich nach dem Verinnerlichen der Scrum-Philosophie, seinen Entwicklungsprozess zu optimieren. So wird Scrum als Stützrad für das Vorankommen in der agilen Entwicklung verwendet, bis der eigene Prozess so etabliert ist, dass man auf die strikte Einhaltung verzichten kann. Da sich Scrum etabliert hat und als Hilfsmittel zur Entwicklung der eigenen Prozesse verwendet werden kann, wird es für dieses Framework ausgewählt. Das Framework darf nicht als vollständig, sondern als Mindestmaß angesehen werden. So ist es möglich, es zu erweitern und an die eigenen Prozesse anzupassen.
Das besondere an Scrum ist, dass nur wenige, aber sehr konkrete Vorgaben gemacht werden. Die Mitarbeiter tragen viel Verantwortung und müssen sich disziplinieren die anfangs ungewohnte Art der Entwicklung durchzuführen. Bei Scrum wird in Iterationen entwickelt, wobei jede Iteration Analyse, Entwicklung und Evaluation beinhaltet. Eine Iteration wird Sprint genannt, die Zeit, die ein Sprint dauern kann beträgt maximal 30 Tage. Die Dauer des Sprints ist die Timebox. Die Größe der Timebox wird bei Projektinitialisierung einmal festgelegt und danach nicht mehr geändert. Für einen Sprint werden Features ausgewählt, die in dieser entwickelt und implementiert werden sollen, am Ende gelten angefangene und nicht entwickelte Features als nicht erfüllt. Am Ende jedes Sprints folgt eine Retrospektive, in der die Mitarbeiter beispielsweise erklären, was gut bzw. schlecht gelaufen ist. Diese Erkenntnisse fließen dann in den nächsten Sprint mit ein und sollen so die Effektivität und die Effizienz steigern. Täglich wird ein Daily-Scrum durchgeführt, kurzes Meeting, in dem die Mitarbeiter ihren aktuellen Status oder Bedürfnisse der Gruppe mitteilen können. So sollen Probleme frühzeitig erkannt und behoben werden.

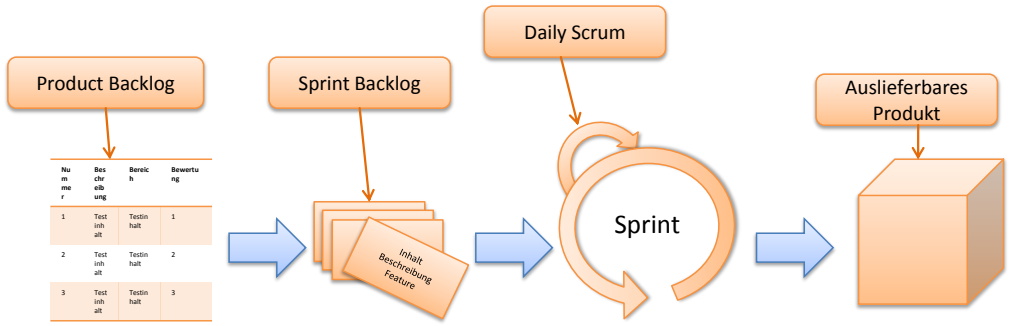

Abbildung 5.1: Ablauf eines Scrum Projektes (Zeichnung nach Vorlage von Roman Pichler (Vgl. [Pic08], S.7))

5.1 Rollen

5.1.1 Product Owner

Der Product Owner ist der Vertreter des Produktes und direkter Ansprechpartner des Teams und des Scrum Masters bezüglich produktbezogener Fragen. Der Product Owner definiert die Anforderungen an die Software und erstellt und verwaltet das Product Backlog. Er arbeitet eng mit dem Team zusammen und vertritt und kommuniziert die Bedürfnisse des Kunden und muss in der Lage sein den Mehrwert zu erkennen. Zu seinen Aufgaben gehört die regelmäßige Abstimmung mit dem Kunden und dem Projektteam. Er erstellt das Product Backlog und verwaltet dieses. Er fügt seine Features ein, ändert Prioritäten und ändert den Status von implementierten Features. Der Product Owner hat durch seine Vertretung des Kunden eine hohe Verantwortung.

5.1.2 Scrum Master

Der Scrum Master ist eine Hilfestellung zur Etablierung von Scrum Prozessen und ist im Optimalfall arbeitslos. Er gibt Hilfestellung bei der korrekten Durchführung von Meetings, Sprints, Retrospektiven und beim Erstellen des Product oder Sprint Backlogs. Er nimmt keine moderierende, sondern beratende Position ein. Mit der Zeit sollte auf seine Hilfe verzichtet werden können.

5.1.3 Team

„Das Team führt alle Arbeiten aus, die zur Umsetzung der Anforderungen in auslieferbare Produktinkremente notwendig sind. Die Softwareentwicklung in Scrum ist somit teambasiert.“([Pic08], S.13) Das Team in einem Scrum Projekt muss sorgfältig ausgewählt werden, so dass alle nötigen Aufgaben gelöst werden können. Es ist wichtig, dass das Team gut zusammenarbeiten kann und es bereit dazu ist, sich selbst zu organisieren und mehr Verantwortung zu tragen als in konservativeren Vorgehensmodellen. Es ist auch möglich, dass sich Mitarbeiter freiwillig für ein Projekt melden. Durch das persönliche Interesse das dadurch gezeigt wird, ist eine höhere Motivation zu erwarten, als wenn ein Mitarbeiter einem Projekt zugeteilt wird.
Das Scrum-Team hat einige Bevollmächtigungen: Es entscheidet selbst wie viele Anforderungen innerhalb der nächsten Sprints in ein Produktinkrement umgewandelt werden können. Es legt somit fest, wie viel Arbeit das Team zuverlässig erledigen kann. Das Team bestimmt die nötigen Arbeitsschritte und organisiert diese selbst. Ebenso sollte ein eingespieltes Team „nein“ zu zu vielen Anforderungen sagen, wenn abzusehen ist, dass die Ziele in einem Sprint nicht erreicht werden können.
“Ein Scrum-Team ist autonom: Das Team muss in der Lage sein, das Sprint-Ziel ohne

wesentliche externe Abhängigkeiten zu erreichen. Ansonsten fällt es dem Team schwer, eine echte Verpflichtung zur Erreichung des Sprint-Ziels abzugeben und auslieferbare Produktinkremente zuverlässig am Ende jedes Sprints zu erstellen.“([Pic08], S.14) Ein Indiz für das nicht autonome arbeiten ist, wenn regelmäßig Aufgaben im Backlog ausserhalb des Teams gelöst werden müssen.
Ein Scrum-Team ist interdisziplinär besetzt, so müssen alle Rollen, die für die Projekt- und Sprint-Ziele benötigt werden, im Team vertreten sein. Beispiele hierfür sind Architekt, Entwickler, Tester, Dokumentationsexperte, Interface Designer, Datenbankexpterte, Konfigurationsmanager oder Usability Experte. „Damit ein interdisziplinäres Team gut zusammenarbeitet, ist es notwendig, dass die Teammitglieder über gemeinsames Softwareentwicklungswissen verfügen und verschiedene Aufgaben im Team wahrnehmen können.“([Pic08], S.15) Trotz Spezialisierung der einzelnen Mitarbeiter müssen diese in der Lage sein, im Team zu arbeiten und bereit sein, auch spezialisierungsfremde Aufgaben zu übernehmen.
Die Teamgröße bei Scrum-Projekten sollte zwischen fünf und neun Mitgliedern sein und ist somit relativ klein. Durch die kleineren Teams wird die Kommunikation untereinander erleichtert und die Einschätzung der einzelnen Personen und deren Leistung vereinfacht. Kleinere Teams sind möglich, jedoch muss dort besonders darauf geachtet werden, dass das Team fachlich in der Lage ist, die Anforderungen zu erfüllen.

5.2 Prozesse

5.2.1 Product Backlog erstellen

Das Product Backlog ist das zentrale Mittel zum Erfassen und Managen von Anforderungen in Scrum. Alle bekannten Anforderungen und Arbeitsergebnisse, die zur Zielerreichung benötigt werden, werden darin verwaltet. Funktionale, non funktionale wie auch Anforderungen an die Benutzerschnittstelle werden dort festgehalten. Ebenso können Tests oder zu beseitigende Bugs enthalten sein. Das Product Backlog wird bei der Initialisierung des Scrum-Projektes vom Product Owner erstellt und im Projektverlauf von diesem gepflegt. Die Aktivitäten, die benötigt werden, werden vom Team für jeden Sprint im Sprint Backlog und nicht im Product Backlog definiert. Das Product Backlog wächst mit der Dauer des Projektes, so können Anforderungen hinzugefügt, detailliert, geändert oder auch gelöscht werden. Damit unterscheidet sich ein Product Backlog vom klassischen Pflichtenheft. Im Regelfall sind die Anforderungen erst grob und werden später erst detailliert formuliert. Zu jeder Anforderung gehört eine Priorität, die darüber Auskunft gibt, wie wichtig die Anforderung für die Zielerreichung ist. Die Priorität kann ebenfalls geändert werden.

5.2.2 Sprint Backlog erstellen

Im Sprint Backlog werden die zu erfüllenden Anforderungen für den nächsten Sprint von dem Team ausgewählt und verwaltet. Es beinhaltet alle Aktivitäten des Teams und fungiert als kollektives Zeitmanagementsystem. Alle Aktivitäten sind in Personenstunden abgeschätzt und sollten möglichst detailliert und präzise beschrieben sein. Im Sprint Backlog wird festgehalten, welche Aufgaben noch erledigt werden müssen, welche in Arbeit und welche abgeschlossen sind. Neben dem aktuellen Status der Aufgabe muss der verantwortliche Mitarbeiter festgehalten werden. Wenn jemand eine Aufgabe übernimmt, setzt er den Status der Aufgabe auf „in Bearbeitung“ und schreibt sein Kürzel hinzu, so dass nachvollzogen werden kann wer an welcher Aufgabe arbeitet.

5.2.3 Daily-Scrum

„Die Daily Scrum ist eine auf 15 Minuten beschränkte Besprechung, die an jedem Arbeitstag am selben Ort zur selben Zeit stattfindet.“([Pic08], S.104) Die Daily Scrum kann zu jeder beliebigen Uhrzeit stattfinden, es wird jedoch von einigen morgens bevorzugt, um den Tag besser planen zu können. Ziel der Daily Scrums ist es, Hindernisse frühzeitig zu erkennen, um eine rechtzeitige Beseitigung zu ermöglichen. Neben den Teammitgliedern muss der Scrum Master und vorzugsweise der Product Owner anwesend sein. Zusätzlich ist es möglich, dass weitere Interessensvertreter beiwohnen, jedoch nur als Zuhörer.
Jedes Mitglied soll im Daily Scrum folgende drei Fragen beantworten:

- Welche Aktivitäten habe ich seit der letzten Daily Scrum abgeschlossen?
- Woran plane ich bis zur nächsten Daily Scrum zu arbeiten?
- Werde ich in irgendeiner Form an der Ausführung einer Aktivität behindert?

Der Scrum Master soll das Meeting leiten und muss darauf achten, dass die 15 Minuten nicht überzogen werden. Die Daily Scrum dient nicht dazu Probleme zu lösen, sondern diese aufzuzeigen.

5.2.4 Sprint Review

Das Sprint Review findet am Ende eines Sprints statt und dauert ein bis zwei Stunden. Je nach Art der Software und Infrastruktur kann das Meeting in einem Besprechungsraum, einem Labor oder im Teamraum stattfinden. Ziel der Sitzung ist es, die Arbeitsergebnisse zu begutachten und den Projektfortschritt transparent zu machen. Der Product Owner prüft, ob die vom Team gesetzten Ziele für den Sprint vollständig erfüllt sind. Teilnehmer des Sprint Reviews sind Product Owner, das Team wie auch der

Scrum Master. Zusätzlich können auch andere Interessensvertreter, wie etwa Vertreter der Benutzer, anwesend sein. Insbesondere die direkte Kommunikation mit Kunde und Benutzern kann sehr hilfreiche Informationen liefern.

5.2.5 Sprint Retrospektive

Die Sprint Retrospektive findet direkt im Anschluss an das Sprint Review statt und dauert, je nach Länge des Sprint Reviews, eineinhalb bis zweieinhalb Stunden. Ziel ist es, die Zusammenarbeit des Teams und die Anwendung der Prozesse zu optimieren. Dabei können Retrospektiven zu einer Steigerung der Produktivität, der Leistungsfähigkeit des Teams und der Softwarequalität führen (Vgl. [Pic08], S.111). Anwesende sind der Product Owner, der Scrum Master und das Team. Zusätzlich ist die Anwesenheit von weiteren Interessensvertretern, wie beispielsweise Führungskräfte, möglich. Diese können bei der Beseitigung von Problemen sehr hilfreich sein.

6 Integration der Standards in Scrum

Da Scrum sich mehr auf das Management als auf die Entwicklung konzentriert, bietet es Freiraum die Softwareentwicklung betreffend. So ist es in der Regel möglich, ein weiteres Vorgehensmodell in ein Scrum Projekt zu integrieren. Oft wird Scrum als Mischform zusammen mit Extreme Programming verwendet (Vgl. [Kar08], S. 24). So lässt sich auch das erweiterte Vorgehensmodell der ISO 13407 in einen Sprint integrieren, um die gewählten Features zu entwickeln. Die Erkenntnisse aus den verschiedenen Prozessen werden als Anforderungen in das Product Backlog geschrieben. Im Product Backlog sind somit alle funktionalen und non-funktionalen Anforderungen an die Software und das System enthalten, die innerhalb der Aktivitäten der Softwareentwickung und des Usability Engineerings herausgearbeitet wurden. Das Vorgehensmodell der ISO 13407 ist die Vorbereitung für die Entwicklung, bei der die Usability Aktivitäten nur bei der Evaluation der fertig entwickelten Komponenten zum Tragen kommen. Die Features für die Entwicklung werden für jeden Sprint aus dem Product Backlog genommen.

7 Framework

Das Framework soll Softwareentwickler, Projektmanager und Projektleiter im Planungs- und Entwicklungsprozess unterstützen, um eine höhere Softwarequalität, insbesondere im Bereich der Benutzbarkeit der Software, zu erreichen. Es soll nicht die Software als Endprodukt, sondern der Prozess der dazu führt, optimiert werden[1]. Es werden Prozesse, deren Ziele und mögliche Methoden und Techniken vorgegeben, um diese zu erreichen. Diese sind allerdings als Mindestanforderung anzusehen. Aus Sicht der Usability Ingenieure ist es wichtig, dass Benutzer ihre Aufgaben effektiv, effizient und zufriedenstellend lösen können. Bei den Softwareentwicklern liegt der Fokus auf dem System, welches korrekt, robust, effizient, wieder verwendbar, wartbar und portabel sein sollte. Als Ausgangspunkt für die Aktivitäten der Entwicklung sind auf der Seite des Usability Engineerings die ISO 13407 als Vorgehensmodell und die ISO PAS 18152 als Definition der Mindestanforderung an Aktivitäten und deren Ziele. Die ISO 12207 sichert die Qualität auf der Seite der Softwareentwicklung. Ähnliche Aktivitäten, jedoch mit verschieden Zielen oder Perspektiven, werden identifiziert und deren Artefakte, sofern sinnvoll, aggregiert um die Informationen zu vervollständigen und die Qualität des Softwareentwicklungsprozesses und damit die Qualität der entwickelten Software zu steigern.
Anstelle eines Frameworks wäre es möglich, das Usability Engineering an eine Firma weiterzugeben. Dies ist auf lange Sicht jedoch teurer und die enge Bindung und dichte der Kommunikation, die zwischen Usability Ingenieuren und Softwareentwicklern bestehen sollte, wird im Regelfall darunter leiden. Bei solch einer Lösung ist die geeignetste Möglichkeit, sich Usability Ingenieure ins Entwicklerteam zu holen die dauerhaft, oder zumindest regelmäßig, sich mit im Büro oder in direkter Nähe befinden, um eine gute Kommunikation zu gewährleisten. Wobei bei externen Teammitgliedern wiederum ein Problem der Akzeptanz aufkommen kann, da Usbility Ingenieure auf die Arbeitsergebnisse der Softwareentwickler achten und so Usability-Probleme aufdecken, was zur Überarbeitung der Software führen kann, was wiederum Entwickler verärgern kann,

[1] Es kann vorkommen, wenn auf Usability keine Rücksicht genommen wurde, teure Überarbeitungen mit mäßiger Qualität im Bereich der Usability Ziele durchgeführt werden müssen, da die nötige Benutzerakzeptanz fehlt. Dies kann beispielsweise an ungewohnten Arbeitsabläufen oder unzureichender Unterstützung liegen. Das Problem ist, wenn die Architektur einmal steht, solche Änderungen mühsam und kostenintensiv sind.

wenn diese der Meinung sind, alles korrekt entwickelt zu haben. Dieses Framework soll von Vornherein sicherstellen, das alle Beteiligten dazu bereit sind, die Konsequenzen eines Usability Engineering Prozesses zu tragen. Um einen positiven Effekt zu spüren, müssen erst einige Hürden genommen und Gewohnheiten abgelegt werden. Jedoch wird sich das Ergebnis positiv von einem Softwareprodukt unterscheiden, welches die Benutzer, deren Bedürfnisse und deren Anforderungen nicht berücksichtigt.

7.1 Begriffsklärung

- **Design Base**
  Die Design Base ist ein Dokument, in dem die grundlegenden Designregeln, Paradigmen und verwendete Patterns festgehalten werden, um Designentscheidungen und die Konsistenz des Designs zu wahren. Wie die Designregeln, Paradigmen und Patterns formuliert werden, bleibt den Entwicklern überlassen. Diese müssen sich jedoch bei Projektbeginn auf einen gemeinsamen Standard einigen und diesen wahren, um die Kommunikation zu vereinfachen. Für die Patterns eignen sich beispielsweise die Design Patterns nach Tidwell [Tid05]. Für die Designregeln können Inhalte aus Styleguides als Beispiel genommen werden.

- **Scrum Master**
  Der Scrum Master hat, neben den Aufgaben die durch Scrum vorgegeben werden, die Verantwortung, dass die Aktivitäten des Usability Engineerings korrekt eingesetzt und durchgeführt werden. Er kennt die möglichen Methoden und Techniken, die für die Zielerreichung der verschiedenen Prozesse eingesetzt werden können und steht beratend zur Seite, wenn diese ausgewählt werden. Der Scrum Master hat die nötige Distanz, um den Entwicklungsprozess zu überwachen und zu bewerten. Er muss in der Lage sein, Fragen, die das Team bezüglich der Durchführung des Usability Engineerings hat, fachlich korrekt zu beantworten. Bei Beginn des Projektes sollte der Scrum Master eng mit dem Team zusammen arbeiten, um einen geregelten Prozess zu etablieren. Je mehr das Team den Usability Engineering Prozess verinnerlicht hat, desto mehr Distanz kann der Scrum Master zu dem Projekt halten. Dies sollte immer angestrebt werden.

- **Product Owner**
  Der Product Owner vertritt die Bedürfnisse des Kunden und der Endanwender. Er kennt die organisatorische Struktur, die Anwender, den Nutzungskontext und die Umgebung in der das zu entwickelnde System eingesetzt werden soll. Er ist so in der Lage, Fragen bezüglich der Anwender zu beantworten und stellvertretend für diese Tests durchzuführen, um die Gebrauchstauglichkeit zu überprüfen. Auch

wenn der Product Owner die Anwender und deren Eigenschaften kennt, darf er nicht der einzige Tester sein. Um die Gebrauchstauglichkeit zu prüfen, müssen die Endanwender das System persönlich testen. Der Product Owner sollte die Anwendungsfälle, in Absprache mit dem Kunden, für Labor- oder Produktivtests entwickeln.

7.2 Vorgehensmodell

Als Grundvoraussetzung für die Anwendung des Frameworks (siehe Abbildung 7.1) wird eine einheitliche Zustimmung zur Durchführung von Usability Engineering aller Beteiligten benötigt. Das Product Backlog wird mit Anforderungen gefüllt, die sich aus der Kontext- und Anforderungsanalayse ergeben. In der Anforderungsanalyse sind alle Anforderungen aus der Softwareentwicklung (Systemanforderungen, Architekturanforderung etc.) und in der Kontextanalyse die Systemanforderungen aus dem Usability Engineering (Anforderungen der Benutzer, Arbeitsumgebung etc.) emthalten. An dieser Stelle ist es schon möglich, Styleguides zu entwickeln und Paradigmen oder Patterns zu definieren. Ob dies möglich ist, hängt stark von der Granulatität des Product Backlogs ab. Nach dem Erstellen des Product Backlogs kann der erste Sprint gestartet werden. Für jeden Sprint wird ein Sprint Backlog erstellt, welches mit Anforderungen oder Features aus dem Product Backlog gefüllt wird. Den Inhalt des Sprint Backlogs gibt der Product Owner, als Vertreter des Kunden, vor. Ist der Sprint Backlog erstellt und der Sprint gestartet, darf dieses nicht mehr verändert werden. Jedoch können Ergebnisse aus dem Entwicklungsprozess das Product Backlog und die Design Base beeinflussen. Jederzeit können neue Einträge gemacht oder weiter aktualisiert werden. Es muss immer darauf geachtet werden, wie sich neue Einträge oder Änderungen auf schon vorhandene Komponenten auswirken. Der Ablauf des Sprints ist bis zur Evaluation gemäß der ISO 13407 durchzuführen. Ergibt die Evaluation der Prototypen keine Mängel, können die Komponenten entwickelt werden. Die entwickelten Komponenten werden produktiv getestet und evaluiert. Ergeben die Tests der entwickelten Komponenten keine Mängel, kann mit einem neuen Sprint Backlog fortgefahren werden. Werden jedoch Mängel gefunden, wird, wie bei Fehlschlag der Evaluation der Protptypen, wieder bei der Kontextanalyse angefangen. Dies ist Notwendig, da sichergestellt werden muss, dass alle Informationen aus der Kontextanalyse korrekt sind und sich die Anforderungen nicht geändert haben.
Wurden alle Features aus dem Product Backlog implementiert oder sind nur noch optionale Features vorhanden und der Kunde wünscht eine Beendigung der Entwicklung, wird eine ausführliche finale Evaluation durchgeführt. Diese sollte umfangreiche Pro-

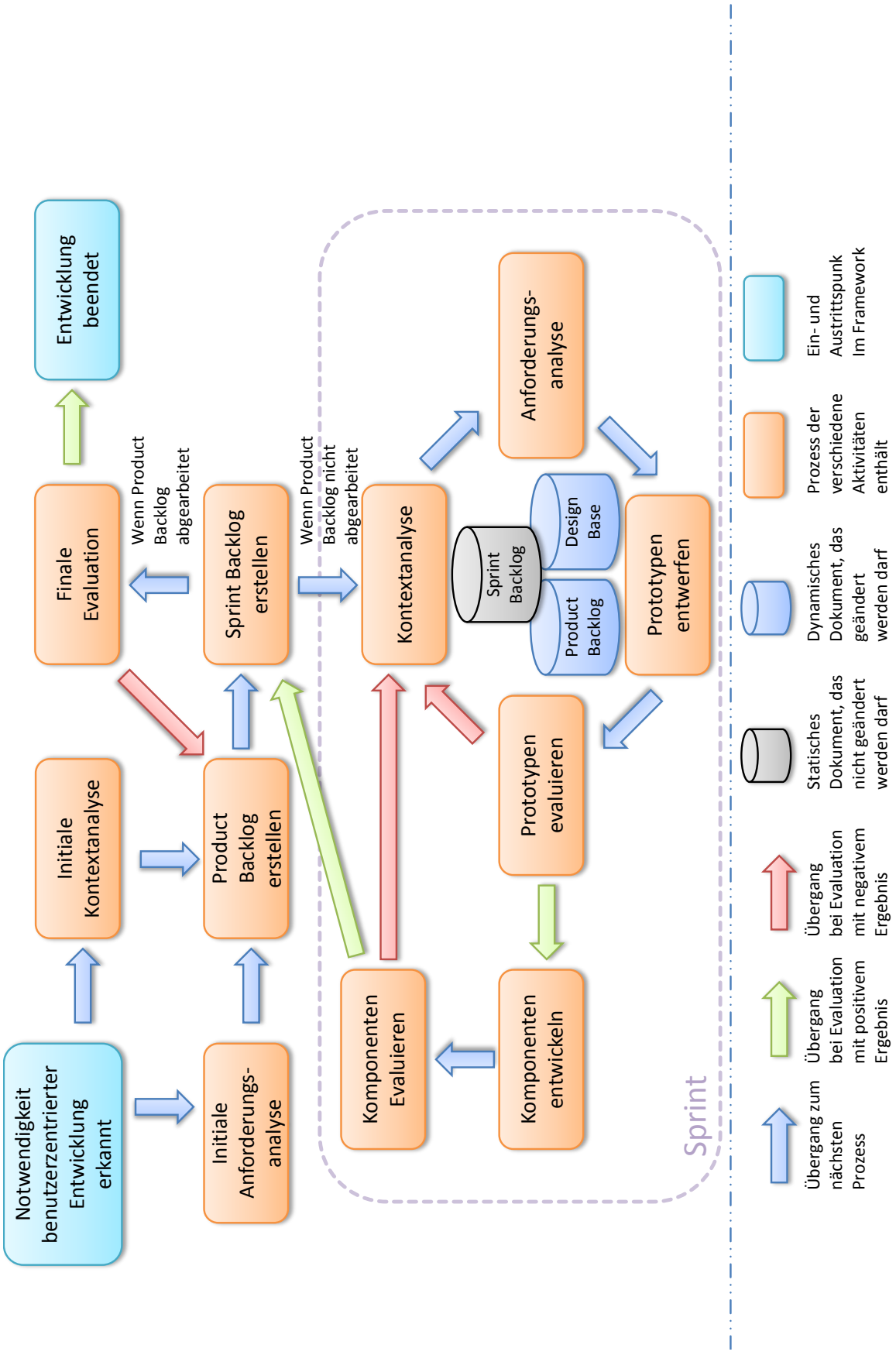

Abbildung 7.1: Vorgehensmodell des Frameworks

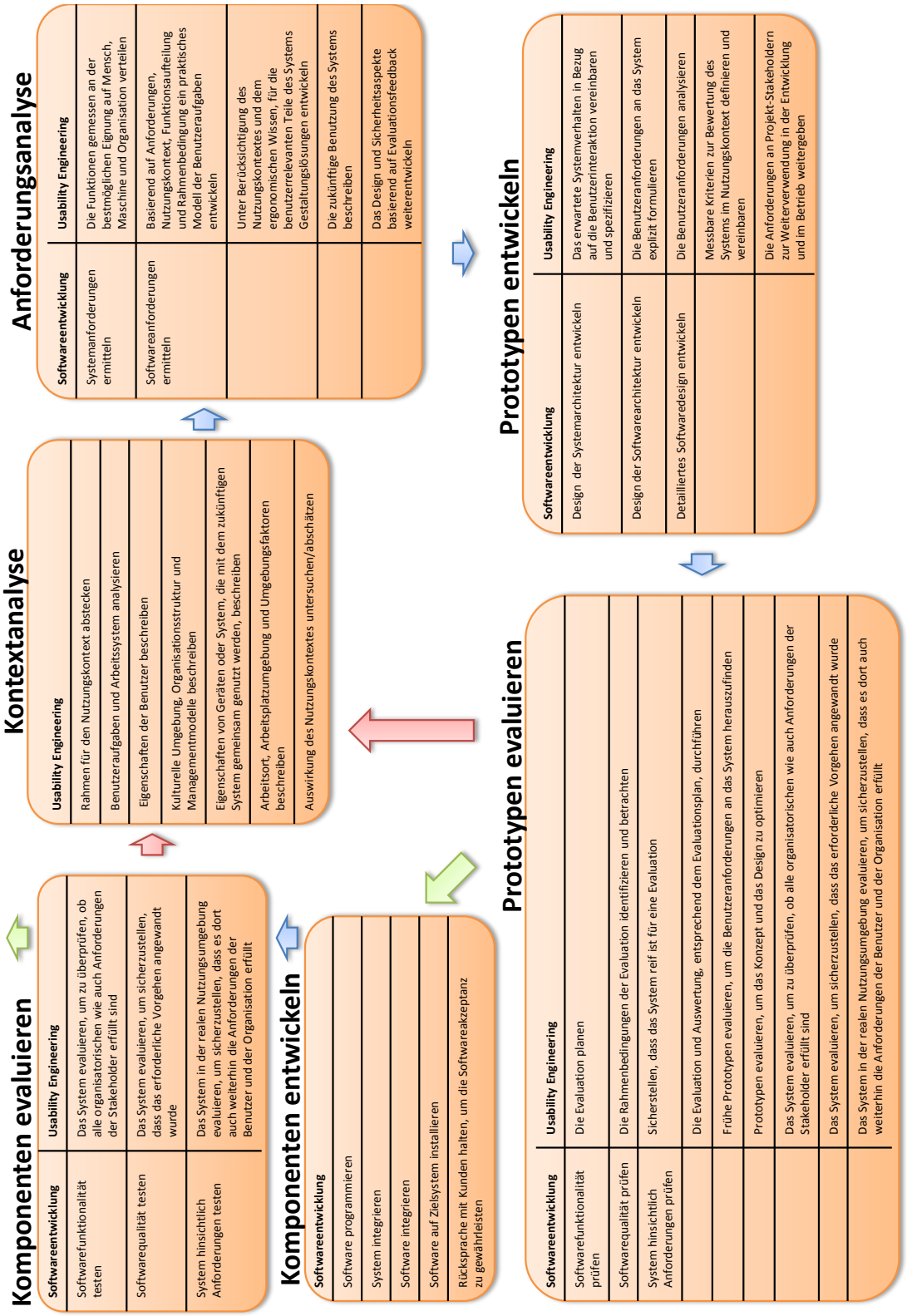

Abbildung 7.2: Das Vorgehensmodell im Detail innerhalb eines Sprints mit den verschiedenen Aktivitäten von Softwareentwicklung und Usability Engineering

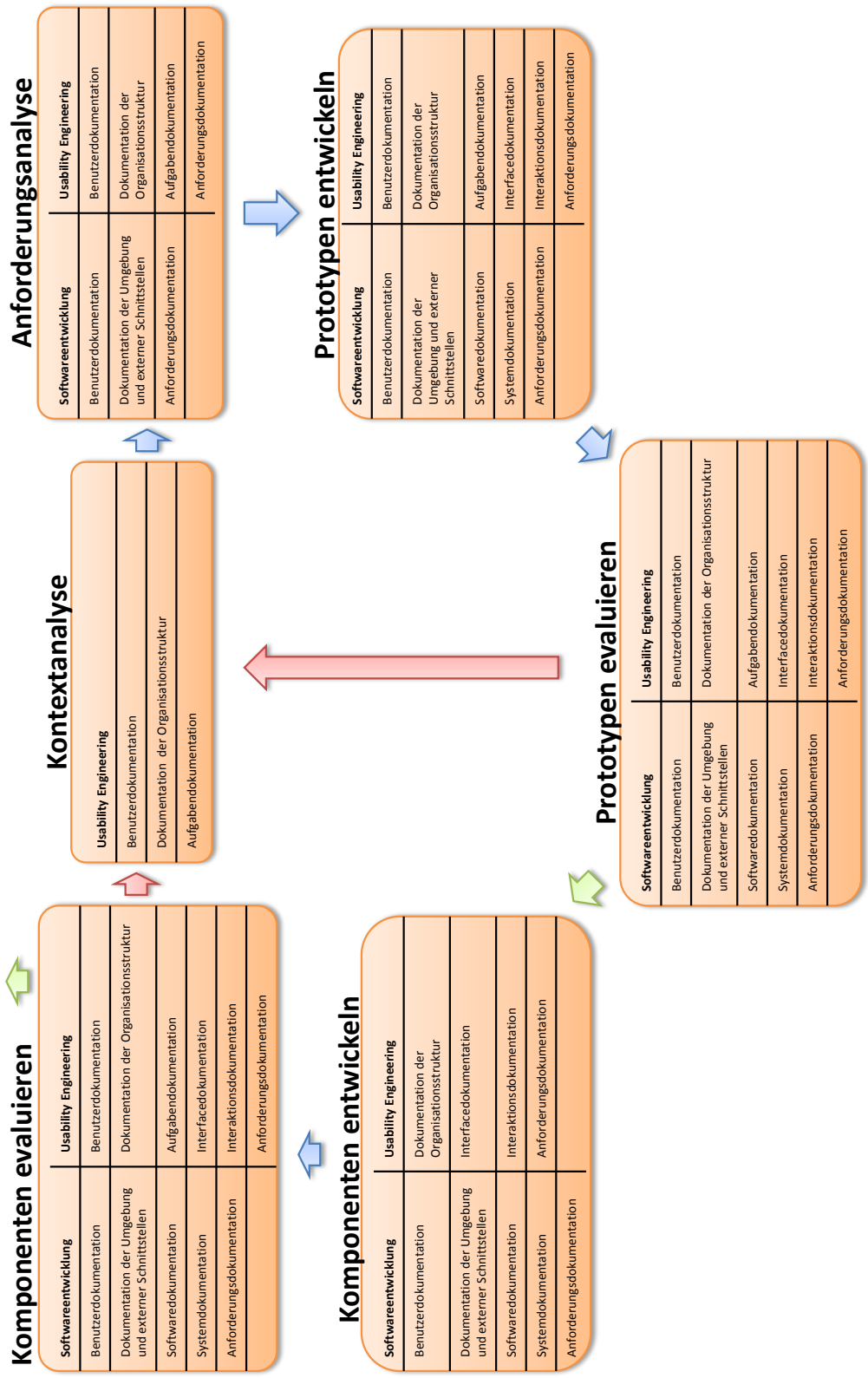

Abbildung 7.3: Das Vorgehensmodell im Detail innerhalb eines Sprints mit den verschiedenen Projekt-Dokumenten von Softwareentwicklung und Usability Engineering

duktivtests mit Benutzern beinhalten. Ergibt die Evaluation keine Mängel, kann das Projekt abgeschlossen werden. Werden Mängel entdeckt, werden diese als Anforderungen formuliert und ins Product Backlog geschrieben, um daraus weitere Sprints starten zu können und um die Software auf den gewünschtes Stand zu bringen.

7.3 Integration der ISO 13407

Die Entwicklungsphasen der ISO 13407 finden sich in jedem Sprint wieder. Jeder Sprint ist eine in sich geschlossene Entwicklung bestimmter Features, diese werden, getrennt von den anderen Sprints, einzeln designed und evaluiert. Um Konsistenz zu wahren und den Designprozess zu unterstützen, werden grundlegende Designregeln, Patterns oder Paradigmen in der Design Base festgehalten. Die Hauptkategorien der ISO 13407 (Kontextanalyse, Anforderungsanalyse, Prototypen entwickeln, Prototypen evaluierten) sind bei der klassischen Softwareentwicklung und bei Usability Engineering identisch, daher werden diese beibehalten. Die Aktivitäten, die in diesen Kategorien ausgeführt werden, um die Ziele dieser zu erreichen, können jedoch unterschiedlich sein. In der Regel ist der Fokus ein Anderer. Bei der Softwareentwicklung liegt er auf dem System und beim Usability Engineering auf dem Benutzer und der Benutzung. Die Kommunikation zwischen beiden Domänen muss nach der Durchführung der einzelnen Prozesse über erstellte Artefakte erfolgen. Dafür muss bekannt sein, was ein Artefakt beinhaltet, wie es entstanden ist und wobei es bei der weiteren Entwicklung helfen soll.

7.3.1 Notwendigkeit von benutzerzentrierter Entwicklung erkennen

Für die Verwendung dieses Frameworks müssen alle beteiligten Personen von der Notwendigkeit benutzerzentriert zu entwickeln überzeugt sein. Die Beteiligten müssen bereit sein, die Entwickler angemessen zu unterstützen, dies bezieht sich beispielsweise auf engen Kontakt mit Benutzern oder ausführliche Tests der entwickelten Features.

7.3.2 Kontextanalyse

Die Kontextanalyse sollte bei Projektstart durchgeführt und zu Beginn eines jeden Sprints nochmals verifiziert werden. Da jederzeit neue Features zum Product Backlog hinzugefügt oder alte geändert oder gelöscht werden können, ist es umso wichtiger zu prüfen, ob der herausgearbeitete Kontext noch korrekt ist. Erkenntnisse aus der Analyse des Kontextes die sich in Anforderungen formulieren lassen, sollen im Product Backlog festgehalten werden.

Aktivitäten des Usability Engineerings

- Rahmen für Nutzungskontext abstecken
- Benutzeraufgaben und Arbeitssystem analysieren
- Eigenschaften der Benutzer beschreiben
- Kulturelle Umgebung, Organisationsstruktur und Managementmodelle beschreiben
- Eigenschaften von Geräten oder System, die mit dem zukünftigen System gemeinsam genutzt werden, beschreiben
- Arbeitsort, Arbeitsplatzumgebung und Umgebungsfaktoren beschreiben
- Auswirkung des Nutzungskontextes untersuchen/abschätzen

7.3.3 Anforderungsanalyse

Die Anforderungsanalyse sollte bei Projektstart durchgeführt und zu Beginn eines jeden Sprints nochmals verifiziert werden. Da jederzeit neue Features zum Product Backlog hinzugefügt oder alte geändert oder gelöscht werden können, ist es umso wichtiger zu prüfen, ob die herausgearbeiteten Anforderungen noch korrekt sind. Geprüft werden müssen die Anforderungen an die einzelnen Features, die in dem Sprint entwickelt werden sollen. Alle Anforderungen sollen im Product Backlog festgehalten werden.

Aktivitäten der Softwareentwicklung

- Softwareanforderungen ermitteln
- Systemanforderungen ermitteln

Aktivitäten des Usability Engineerings

- Die Funktionen gemessen an der bestmöglichen Eignung auf Mensch, Maschine und Organisation verteilen
- Basierend auf Anforderungen, Nutzungskontext, Funktionsaufteilung und Rahmenbedingung ein praktisches Modell der Benutzeraufgaben entwickeln
- Unter Berücksichtigung des Nutzungskontextes und dem ergonomischen Wissen für die benutzerrelevanten Teile des Systems Gestaltungslösungen entwickeln
- Die zukünftige Benutzung des Systems beschreiben

- Das Design und Sicherheitsaspekte basierend auf Evaluationsfeedback weiterentwickeln

7.3.4 Gestaltungslösungen entwickeln

In jedem Sprint werden für die ausgewählten Features Gestaltungslösungen entwickelt. Bei der Entwicklung soll darauf geachtet werden, dass die Inhalte aus dem Design Base berücksichtigt werden. Ebenso sollen neue Patterns, Designregeln, oder Paradigmen hinzugefügt werden, falls sie in dem Sprint verwendet werden. Es ist darauf zu achten, dass nicht nur ein Entwurf, sondern mindestens drei entwickelt werden. Eine zu geringe Anzahl von Prototypen schränkt die Perspektive auf den Lösungsraum zu sehr ein. Erkennbar unterschiedliche Prototypen ermöglichen Diskussionen, aus denen neue Ideen oder Verbesserungen hervorgehen können. Dabei ist es hilfreich, erst Lo-Fi[2] Prototypen zu entwickeln, da diese weniger zeitintensiv sind und es leichter fällt, diese zu verwerfen, wenn man nicht einen ganzen Arbeitstag darauf verwendet hat.

Aktivitäten der Softwareentwicklung

- Design für die Systemarchitektur entwickeln
- Architektur-Design für die Software entwickeln
- Detailliertes Design für die Software entwickeln

Aktivitäten des Usability Engineerings

- Das erwartete Systemverhalten in Bezug auf die Benutzerinteraktion vereinbaren und spezifizieren
- Die Benutzeranforderungen an das System explizit formulieren
- Die Benutzeranforderungen analysieren
- Messbare Kriterien zur Bewertung des Systems im Nutzungskontext definieren und vereinbaren
- Die Anforderungen an Projekt-Stakeholder zur Weiterverwendung in der Entwicklung und im Betrieb weitergeben

[2]Lo-Fi steht für low fidelity und beschreibt in diesem Kontext Prototypen, die nicht komplett ausgearbeitet sind, wobei es eher um die Idee geht. Lo-Fi Prototypen sind in der Regel nicht interaktiv und die Farbgebung oder Position einzelner Elemente ist nicht zwingend final.

7.3.5 Gestaltungslösungen evaluieren

Jeder entwickelte Prototyp muss auf verschiedenen Wegen evaluiert werden. Heuristiken sollen grundlegende Designfehler aufdecken, wie beispielsweise einen zu hohen cognitive load. Gespräche, Fragebögen und cognitive walkthroughs sollen Probleme bei der Benutzbarkeit der Features finden. Erkenntnisse aus der Evaluation, die auch andere Features betreffen, werden im Design Base festgehalten.

Aktivitäten der Softwareentwicklung

- Softwarefunktionalität prüfen
- Softwarequalität testen
- System hinsichtlich der Anforderungen testen

Aktivitäten des Usability Engineerings

- Die Evaluation planen
- Die Rahmenbedingungen der Evaluation identifizieren und betrachten
- Sicherstellen, dass das System reif ist für eine Evaluation
- Die Evaluation und Auswertung, entsprechend dem Evaluationsplan, durchführen
- Frühe Prototypen evaluieren, um die Benutzeranforderungen an das System herauszufinden
- Prototypen evaluieren, um das Konzept und das Design zu optimieren
- Das System evaluieren, um zu überprüfen, ob alle organisatorischen wie auch Anforderungen der Stakeholder erfüllt sind
- Das System evaluieren um sicherzustellen, dass das erforderliche Vorgehen angewandt wurde
- Das System in der realen Nutzungsumgebung evaluieren, um sicherzustellen, dass es dort auch weiterhin die Anforderungen der Benutzer und der Organisation erfüllt

7.3.6 Komponenten entwickeln

Die Features werden nach den Anforderungen und den Erkenntnissen aus der Evaluation, orientiert an den Informationen aus dem Design Base, entwickelt.

Aktivitäten der Softwareentwicklung

- Software programmieren
- Software integrieren
- System integrieren
- Software auf dem Zielsystem installieren
- Rücksprache mit Kunden halten (evtl. bei Schulungen unterstützen), um die Softwareakzeptanz zu gewährleisten

7.3.7 Komponenten evaluieren

Die Features werden nach den Anforderungen und den Erkenntnissen aus der Evaluation, orientiert an den Informationen aus der Design Base, entwickelt.

Aktivitäten der Softwareentwicklung

- Softwarequalitätstests durchführen

Aktivitäten des Usability Engineerings

- Das System evaluieren, um zu überprüfen, ob alle organisatorischen wie auch Anforderungen der Stakeholder erfüllt sind
- Das System evaluieren, um sicherzustellen, dass das erforderliche Vorgehen angewandt wurde
- Das System in der realen Nutzungsumgebung evaluieren, um sicherzustellen, dass es dort auch weiterhin die Anforderungen der Benutzer und der Organisation erfüllt

7.4 Methoden und Techniken

7.4.1 Grundlegendes

Benutzer

Wie der Begriff „benutzerzentrierte Gestaltung“ schon sagt, sind die Benutzer, die das zu entwickelnde System verwenden werden, Ausgangspunkt für die Modellierung. Daher sind diese ein stetiger Bestandteil der Analyse, dem Prototyping, der Evaluation und der Entwicklung. Da dies gerade bei der Entwicklung schnell in Vergessenheit gerät,

kann beispielsweise neben dem Poster des Datenbankmodells und der Architektur, eine Übersicht über Key-User und deren Aufgaben im Büro aufgehangen werden.

Kommunikation

Es muss viel kommuniziert werden im Kreis aller Beteiligten. Jedes Gespräch kann neue Erkenntnisse bringen und hilft den Gesprächspartner zu verstehen und seine Gedanken nachzuvollziehen. Dies ist besonders wichtig bei Gesprächen mit den Benutzern, allerdings ist dies auch wichtig, wenn es um eine gute Zusammenarbeit an einem Projekt geht. Es ist sehr wichtig, dass neue Erkenntnisse sofort festgehalten werden, zusätzlich muss diese Dokumentation der Erkenntnisse mit dem Gesprächspartner geprüft werden, damit Missverständnisse, die durchaus häufig vorkommen, besonders wenn man sich in einer zuvor fremden Domäne bewegt, ausgeschlossen werden.

Artefakte

Bei der Erstellung verschiedener Artefakte muss sich das Projektteam vorher auf eine einheitliche Struktur geeinigt haben. So sind diese besser vergleichbar und jedes Projektmitglied weiß, wie ein gewisses Artefakt aussieht, auch wenn es ein Anderer entwickelt hat und muss sich nicht erst in die Struktur des Anderen einarbeiten. Wenn Artefakte aus einem Softwareentwicklungsprozess oder aus einem Usability Engineering Prozess entstehen, wozu ein adäquates Gegenstück in der anderen Domäne existiert, können diese Artefakte, sofern sinnvoll, aggregiert werden, um die relevanten Informationen zu einem Thema nicht außer Acht zu lassen.

7.4.2 Kontext- und Anforderungsanalyse

Interviews durchführen

Lockere Interviews (keine Verhöre!) sind eine der effektivsten Möglichkeiten, Anforderungen zu ermitteln (Vgl. [Ben05], S. 216). Hierbei kann besonders auf die individuellen Bedürfnisse des Einzelnen eingegangen werden. Durch so ein Gespräch können auch Anforderungen hinterfragt werden. Dies ist wichtig, denn oft sind die Anforderungen, die ein Benutzer oder anderer Stakeholder formuliert, nicht zwingend dass, was er benötigt. Beispielsweise würden viele Personen die jeden Morgen auf dem Weg zur Arbeit im Stau stehen, besser ausgebaute Straßen fordern, wobei es vermutlich um eine Möglichkeit geht, seinen Arbeitsplatz zeitnah und ohne großen Aufwand zu erreichen. Aus der zweiten Formulierung sind dann auch gut vernetzte öffentliche Verkehrsmittel eine Option.

Fragebögen verwenden

Fragebögen sind eine Analysemethode, die immer angewandt werden kann. Es gibt viele vorgefertigte Fragebögen, die in verschiedenen Projektphasen und Schwerpunkten verwendet werden können. Man unterscheidet hierbei zwischen qualitativen und quantitativen Fragen. Bei qualitativen Fragen handelt es sich um Freitextantworten, wobei es sich bei quantitativen Fragen um multiple choice Antworten zum Ankreuzen handelt. Es sollten vorgefertigte Fragebögen, wie beispielsweise der Fragebogen der ISO 9241-110 oder SUMI, verwendet werden, da die Entwicklung eines Fragebogens eine sehr schwierige Aufgabe ist, auch wenn dies auf den ersten Blick nicht so scheint.

Beobachten

Neben den Interviews und Fragebögen sind Beobachtungen ein weiteres Mittel, was gerade bei unbewusstem Handeln zum Tragen kommt. Einige Informationen erscheinen den Benutzern nicht als wichtig, da sie für diese alltäglich sind, es ihnen einfach nicht bewusst ist, oder sie es einfach vergessen haben zu erwähnen. Für die Entwickler können gerade solche Informationen entscheidend für Anforderungen und die Modellierung des Interfaces sein, da gerade unbewusstes Handeln zu Problemen führen kann, wenn nicht die erwarteten Reaktionen auftreten. Daher sollten nach Möglichkeit Benutzer in ihrer natürlichen Umgebung beobachtet werden.

Stakeholder ermitteln

Stakeholder sind alle Personen die ein Interesse an dem Projekt haben. Dies sind beispielsweise Projekt- und Abteilungsleiter, wie auch Mitarbeiter oder Benutzer.

User Profiles erstellen

User Profiles sind narrative Beschreibungen der Eigenschaften eines Benutzer. Sinn und Zweck ist es, die Merkmale der Benutzer nicht in Tabellen, sondern in einer Geschichte unterzubringen, da die narrative Form gehirngerechter ist und die Benutzer zugleich menschlicher dargestellt werden, als bei einer tabellarischen Form. Headerinformationen dienen lediglich zu Verwaltungszwecken, so ist es möglich, Benutzer die gewissen Kriterien entsprechen, schnell zu finden. Als Headerinformationen sollten Name, Geschlecht und Alter als Mindestanforderungen gegeben sein. Je nach Situation können beispielsweise auch Informationen über Berufsausbildung oder Berufserfahrung im Kopfbereich des Dokumentes festgehalten werden. Vorzugsweise sollten die User Profiles mit einem Foto versehen werden. Durch den zusätzlichen visuellen Reiz des Fotos bleiben die Ei-

genschaften der Benutzer besser abrufbar. Zusätzlich steigt die Akzeptanz gegenüber der fiktiven Person, wenn sie ein menschliches Gesicht bekommt.

Personaes erstellen

Personae sind fiktive Geschichten zu Personen. Personae sind aufgebaut wie User Profiles, beziehen sich allerdings nicht auf reale, sondern auf fiktive Benutzer. Personae können zum Einsatz kommen, wenn keine realen Benutzer verfügbar sind, oder wenn der Aufwand, beispielsweise finanzieller Natur, dafür zu groß ist.

Hierarchical Task Analysis

Die Hierarchical Task Analysis (kurz HTA) ist eine Methode, mit der sich Aufgaben analysieren und dokumentieren lassen. Die HTA ist gut geeignet, um ein deskriptives Modell[3] eines Systems zu entwickeln. Die Qualität einer HTA hängt stark von der Granularität ab, in der die HTA dokumentiert wird. Da es keine Vorgaben gibt und auch die Formulierungen offen sind, kann die Qualität stark variieren. Daher sollten HTAs auch nicht für präskriptive Modelle[4] verwendet werden.
Eine HTA beschreibt den Weg zu einem Ziel (goal) und ist, wie der Name schon sagt, hierarchisch organisiert. Die Hierarchie ist wie folgt:

- **goal**
  Zielbeschreibung, was das Ziel der Aufgabe ist.
- **sub goal**
  Teilziele die zur Zielerreichung benötigt werden.
- **plan** or **action**
  Eine action ist eine sehr eingegrenzte Handlung, beispielsweise „Button drücken“ oder „Schranktür öffnen“. Actions können auch in plans zusammengefasst werden, dies sind dann aufeinander folgende Aktionen.

Die folgende HTA beschreibt, wie die Bezeichnung des goal im obersten Element schon zum kommunizieren versucht, den möglichen Ablauf einer Nachhilfestunde, bei Schülern der 8.-10. Klasse. Die HTA ist relativ fein aufgelöst und könnte bei Bedarf aber noch tiefer ins Detail gehen. Die HTA ist für den Verwendungszweck, die Beschreibung des Ablaufes, bei einer Nachhilfestunde von Schüler der 8.-10. Klasse, angemessen.
Damit klar wird in welcher Reihenfolge die plans und actions ausgeführt werden können, wird neben der hierarchischen Struktur eine Plansequenz benötigt, die Abläufe beschreibt.

[3] Deskriptive Modelle beschreiben den Ist-Zustand eines Systems
[4] Präskriptive Modelle beschreiben den Soll-Zustand eines Systems.

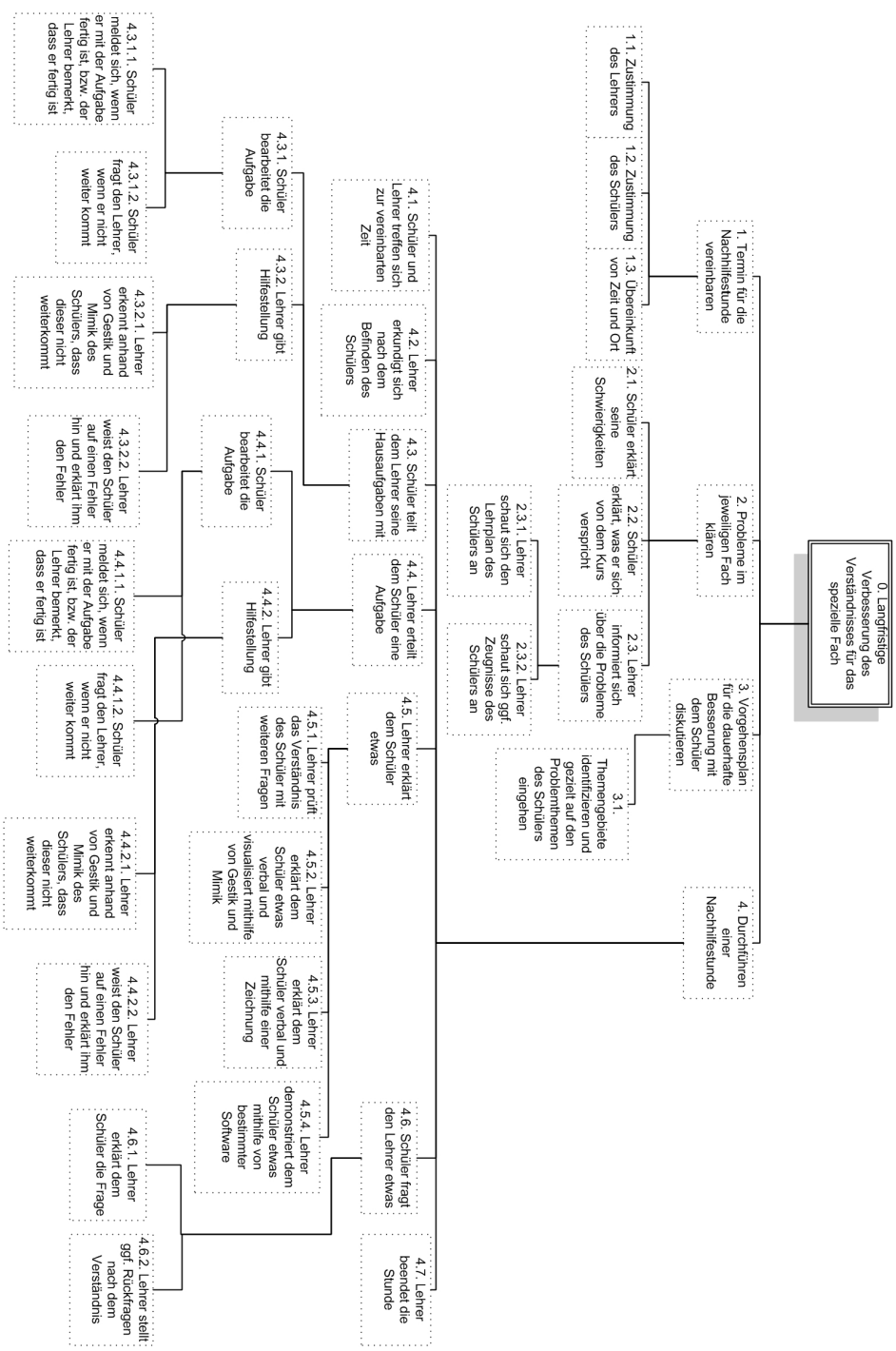

Abbildung 7.4: Beispiel-HTA einer Nachhilfestunde

HTA für Einzelunterricht 8.-10. Klasse

- 0 Langfristige Verbesserung des Verständnisses für das spezielle Fach
- 1 Termin für die Nachhilfestunde vereinbaren
  - 1.1 Zustimmung des Lehrers
  - 1.2 Zustimmung des Schülers
  - 1.3 Übereinkunft von Zeit und Ort
- 2 Probleme im jeweiligen Fach klären
  - 2.1 Schüler erklärt seine Schwierigkeiten
  - 2.2 Schüler erklärt, was er sich von dem Kurs verspricht
  - 2.3 Lehrer informiert sich über die Probleme des Schülers
    - 2.3.1 Lehrer schaut sich den Lehrplan des Schülers an
    - 2.3.2 Lehrer schaut sich ggf. Zeugnisse des Schülers an
- 3 Vorgehensplan für die dauerhafte Besserung mit dem Schüler diskutieren
  - 3.1 Themengebiete identifizieren und gezielt auf die Problemthemen des Schülers eingehen
- 4 Durchführen einer Nachhilfestunde
  - 4.1 Schüler und Lehrer treffen such zur vereinbarten Zeit
  - 4.2 Lehrer erkundigt sich nach dem Befinden des Schülers
  - 4.3 Schüler teilt dem Lehrer seine Hausaufgaben mit
    - 4.3.1 Schüler bearbeitet die Aufgabe
      - 4.3.1.1 Schüler meldet sich, wenn er mit der Aufgabe fertig ist, bzw. der Lehrer bemerkt, dass er fertig ist
      - 4.3.1.2 Schüler fragt den Lehrer, wenn er nicht weiter kommt
    - 4.3.2 Lehrer gibt Hilfestellung
      - 4.3.2.1 Lehrer erkennt anhand von Gestik und Mimik des Schülers, dass dieser nicht weiter kommt
      - 4.3.2.2 Lehrer weist den Schüler auf einen Fehler hin und erklärt ihm den Fehler
  - 4.4 Lehrer erteilt dem Schüler eine Aufgabe
    - 4.4.1 Schüler bearbeitet die Aufgabe
      - 4.4.1.1 Schüler meldet sich, wenn er mit der Aufgabe fertig ist, bzw. der Lehrer bemerkt, dass er fertig ist
      - 4.4.1.2 Schüler fragt den Lehrer, wenn er nicht weiter kommt
    - 4.4.2 Lehrer gibt Hilfestellung
      - 4.4.2.1 Lehrer erkennt anhand von Gestik und Mimik des Schülers, dass dieser nicht weiter kommt
      - 4.4.2.2 Lehrer weist den Schüler auf einen Fehler hin und erklärt ihm den Fehler
  - 4.5 Lehrer erklärt dem Schüler etwas
    - 4.5.1 Lehrer prüft das Verständnis des Schülers mit weiteren Fragen
    - 4.5.2 Lehrer erklärt dem Schüler etwas verbal und visualisiert mithilfe von Gestik und Mimik
    - 4.5.3 Lehrer erklärt dem Schüler verbal und mithilfe einer Zeichnung
    - 4.5.4 Lehrer demonstriert dem Schüler etwas mithilfe von bestimmter Software
  - 4.6 Schüler fragt den Lehrer etwas
    - 4.6.1 Lehrer erklärt dem Schüler die Frage
    - 4.6.2 Lehrer stellt ggf. Rückfragen nach dem Verständnis
  - 4.7 Lehrer beendet die Stunde

Abbildung 7.5: HTA Sequenz ausformuliert

plan 0 do 1. - 4. in dieser Reihenfolge
plan 1 do 1.1. - 1.2. - 1.3. in beliebiger Reihenfolge
plan 2 do 2.1 – 2.2. - 2.3. in beliebiger Reihenfolge
plan 2.3. in beliebiger Reihenfolge
plan 4 do 4.1 – 4.2. - 4.3. - 4.4. - 4.5. - 4.6. - 4.7. while(time) 4.1. - 4.2. in dieser Reihenfolge, 4.7. am Ende, Rest in beliebiger Reihenfolge
plan 4.3. in dieser Reihenfolge
plan 4.3.1. in beliebiger Reihenfolge, Wiederholung
plan 4.3.2. in beliebiger Reihenfolge, Wiederholung
plan 4.4. in dieser Reihenfolge
plan 4.4.1. in beliebiger Reihenfolge, Wiederholung
plan 4.4.2. in beliebiger Reihenfolge, Wiederholung
plan 4.5. in beliebiger Reihenfolge
plan 4.6. in dieser Reihenfolge, Wiederholung

Abbildung 7.6: HTA Plansequenz

use cases

Es gibt viele Arten von Use Cases die verschiedene Schwerpunkte haben. Use cases sind gute geeignet, um ein präskriptives Modell des Systems zu entwickeln. Folgend werden nur zwei Arten von use cases behandelt, weitere sind beispielsweise concrete use cases oder user stories[5].

essential use cases

Die Besonderheit bei essential use cases liegt in der Technikunabhängigkeit. Essential use cases bestehen aus einer zweispaltigen Matrix, die in der linken Spalte die Intention des Benutzers und in der rechte Spalte die Verantwortlichkeit des System beinhaltet. Es ist bei der Formulierung darauf zu achten, keine Beschreibungen oder typische Komponenten zu verwenden, um sich frei von der Implementierung über das Systemverhalten Gedanken zu machen. Wenn sich beispielsweise ein Benutzer am System anmelden möchte, wäre eine typische Formulierung des Systemverhaltens „Login Maske anzeigen“. Ein gewünschtes Systemverhalten bei einem essential use case wäre allerdings „Authentifizierungsmöglichkeit bieten“. Durch diese Formulierung entsteht nicht schon ein Bild von der gewohnten Login-Maske, wie sie auf jeder Internetseite zu finden ist, sondern lässt alle Möglichkeiten offen. So könnte auch eine Authentifizierung durch einen RFID-Chip, Finger- oder Eyescan möglich sein. Es geht also darum, das gewünschte Systemverhalten zu beschreiben, ohne konkrete Implementierungsgedanken zu haben oder diese bei anderen Lesern zu erwecken.

[5] User stories sind von der Namensgebung her keine use cases, verfolgen aber ein ähnliches Ziel und sind hauptsächlich in der agilen Entwicklung vertreten.

user intention	system responseability
identify self	
	verify identity
	offer choices
choose	
	dispense cash
take cash	

Tabelle 7.1: Beispiel von einem essential use case von der User-Intention an einem Bankautomaten

use cases nach Cockburn

Use Cases nach Alistair Cockburn sind eine verbreitete Methode in der Softwareentwicklung, Anwendungsfälle auszuformulieren. Die Beschreibungen sind im Verhältnis zu essential use cases sehr umfangreich. Enthaltene Daten sind:

- **Use case Name und Identifikationsnummer**
  Jeder use case hat einen Namen und eine eindeutige Nummer, über den er identifiziert werden kann.
- **Akteure** *(actors)*
  Beschreibt welche Akteure an dem Andwendungsfall beteiligt sind.
- **Vorbedingungen** *(precondition)*
  Beschreibt welche Bedingung erfüllt sein müssen, damit der Andwendungsfall erfolgreich ablaufen kann.
- **Hauptablauf** *(main flow)*
  Beschreibt den „normalen“ Ablauf im Anwendungsfall.
- **Alternativer Ablauf** *(alternative flow)*
  Beschreibt einen oder mehrere alternative Ablaufmöglichkeiten.
- **Nachbedingungen** *(postcondition)*
  Beschreibt die Bedinungen, die nach Haupt- oder alternativen Ablauf erfüllt sein müssen, damit der Anwendungsfall als erfolgreich abgeschlossen gilt.
- **Ablauf im Fehlerfall** *(exceptional flow)*
  Beschreibt einen oder mehrere Abläufe im Fehlerfall.

- **Nachbedingung bei Fehlerfall** *(postcondition)*
  Beschreibt die Bedingungen, die nach dem Ablauf im Fehlerfall erfüllt sein müssen.

use case Szenarien

Use case Szenarien sind Instanzen von use cases. Sie sind in narrativer Form geschrieben und beschreiben einen möglichen Ablauf aus dem zugehörigen Anwendungsfall. Use case Szenarien können analytisch, beispielsweise deskriptiv bei Re-Designs, oder auch präskriptiv verwendet werden. Die narrative Form vereinfacht die Kommunikation und die Verarbeitung und Speicherung im Gehirn.

7.4.3 Design und Prototyping

Gestaltgesetze

Durch die gezielte Verwendung von Gestaltgesetzen, können visuelle Effekte so eingesetzt werden, dass diese für den Benutzern unterstützend wirken. Ebenso können durch das Wissen über die Gestaltgesetze negative Einflüsse vermieden werden.

Metaphern

Metaphern beschreiben bildhafte Repräsentationen von abstrakten Konzepten. Metaphern werden sehr oft unbewusst im täglichen Sprachgebrauch, aber auch gezielt verwendet. Der Vorteil von Metaphern ist, dass es bildhafte Repräsentationen sind, die abgerufen und vom working memory schneller verarbeiten werden als auditive. Metaphern sind so mächtiges Werkzeug um Information zu verpacken. Jedoch sind Metaphern stark kulturabhängig. So kann ein Pfeil nach links, als „Gehe zur vorherigen Seite“-Funktion, in Ländern wie Japan, wo von rechts nach links gelesen wird, missverstanden werden. Man unterscheidet folgende Arten von Metaphern:

- bildliche
- sprachliche
- onomatopoetische
- gestische

Design Patterns nach Tidwell

Design Patterns sind ein Gerüst, welches erst an einen Kontext angepasst werden muss, dann aber ein mächtiges Werkzeug bietet, um gebrauchstaugliche Interfaces zu entwickeln. Die einzelnen Pattern sind relativ fein beschrieben, trotzdem allgemein genug, um sie an einen spezifischen Kontext anpassen zu können. Die Pattern Languages[6] sind nicht dafür geeignet (Vgl. [Tid05]), eine Evaluation durchzuführen, sondern stellen eine Richtline bei der Entwicklung von Prototypen und der Implementierung der Software dar. Es werden viele Komponenten oder Interaktionsmöglichkeiten beschrieben. Viele dieser Pattern sind sicherlich schon bekannt, aber es existiert kein konkreter Name. Durch die Namenszuordnung wird die Kommunikation zwischen Projektmitgliedern und die Diskussion bezüglich Prototypen vereinfacht, da keine Beschreibung, beispielsweise „ein zweigeteiltes Panel mit einer Übersichtskarte und einer detaillierten Ansicht“, sondern ein Name verwendet werden kann.
Tidwell hat die Design Pattern in verschiedene Kategorien eingeteilt, die verschiedene Anforderungen an die Software beschreiben. Einzelne Patterns können sich untereinander widersprechen, was daran liegt, dass die Patterns für die Anforderungen ausgewählt werden müssen und diese unterschiedlich sein können.

What Users do

- **Safe Exploration**
  Wenn ein Benutzer die Möglichkeit hat, ein Interface auszuprobieren, ohne zwangsläufig persistente Änderungen hervorzurufen, kann er dadurch gut lernen, wie sich das System verhält. So können neue Wege ausprobiert werden, ohne Stress bezüglich des korrekten Weges haben zu müssen, da Änderungen Rückgängig gemacht, oder abgebrochen werden können.

- **Instant Gratification**
  Benutzer sollen jederzeit ein Ergebnis über ihre Interaktion sehen. Es ist für einen Benutzer eine Erfüllung, wenn er nach wenigen Sekunden der Benutzung ein Erfolgserlebnis hat.

- **Satisficing**
  Interfaces sollten so gestaltet sein, dass sie beim Scannen durch einen Benutzer Lösungsmöglichkeiten für seine Aufgabe bieten. Satisficing bedeutet in diesem Kontext, dass sich der Benutzer denken kann „Ich glaube das ist, was ich brauche!“, ohne es zu Prüfen oder in irgend einer Art und Weise sicher sein zu können.

[6] Pattern Languages bestehen aus Patterns die für den Anwendungsbereich ausgewählt wurden

Die Aufgabe der Designer besteht darin, klare Beschriftungen und Metaphern zu benutzen.

- **Changes in Midstream**
  Falls sich die Ziele eines Benutzers während einer Interaktion ändern, sollte Möglichkeiten geboten werden, den aktuellen Prozess zu beenden und ihn, im optimalen Fall, zu einem späteren Zeitpunkt wieder aufnehmen können.

- **Deferred Choices**
  Oft werden Eingaben durch den Benutzer verlangt. Sofern diese nicht unmittelbar nötig sind, sollten diese auch zu einem späteren Zeitpunkt eingegeben werden können. Solche Eingaben werden beispielsweise oft bei Registrierungen verlangt, wobei die meisten Benutzer die Felder überspringen, oder, wenn eine Eingabe nötig ist, zu der ein Benutzer nicht bereit ist, mit fiktiven Daten füllen.

- **Incrementel Construction**
  Benutzer sollten die Möglichkeit haben, ihre Entscheidungen zu korrigieren um sich inkrementell einer Lösung nähern zu können.

- **Habituation**
  Gewohnte Interaktionsmöglichkeiten oder Metaphern sollten konsistent eingehalten werden. Der Benutzer soll sich nicht wundern, warum beispielsweise das Tastenkürzel zum Speichern des Dokumentes in manchen Masken anders ist.

- **Spatial Memory**
  Oft werden Interaktionsobjekte nicht anhand ihrer Beschriftung, sondern anhand ihrer Position im Interface identifiziert. Daher sollte darauf geachtet werden, dass ähnliche Interaktionsmöglichkeiten an der erwarteten Stelle zu finden sind. Ein klassisches Beispiel ist der „OK“ Button bei Windowsanwendungen.

- **Prospective Memory**
  Menschen markieren sich Gegenstände, um sich an etwas zu erinnern. Beispielweise wird ein Buch auf den Tisch gelegt, um sich daran zu erinnern, dass man es am nächsten Tag wieder abgeben muss. Ein Interface sollte die Möglichkeit bieten, sich selbst eine Erinnerung „basteln“[7] zu können.

- **Streamlined Repitition**
  Oft haben Benutzer das Gefühl, dass sie eine Operation sehr oft wiederholen müssen. Beispielweise beim Kopieren und Ersetzen mehrer Dateien. In diesem Fall

[7] Es wird bewusst ein so unpräzises Wort gewählt, da es nicht auf Schreiben oder Malen festgesetzt werden sollte. Der Benutzer muss selbst entscheiden, welche Metapher für ihn am Besten ist.

könnte eine „Replace All“ Funktion Abhilfe schaffen. Es soll versucht werden, solche Wiederholungen zu vermeiden, oder den Benutzer die Möglichkeit zu geben, eine Standardoperation für den Prozess auszuwählen.

- **Keyboard Only**
  Einige Benutzer arbeiten größtenteils oder ausschließlich über die Tastatur und meiden die Eingabe mit einer Maus. Solche Benutzer sollten in ausreichendem Maße unterstützt werden.

- **Other People's Advice**
  Es muss berücksichtigt werden, dass Benutzer sich Aussagen anderer zu Herzen nehmen. Auch wenn sie dies verneinen würden, werden sie durch Aussagen beeinflusst. Beispielsweise sind Kommentare von Benutzern bei Amazon ein wichtiges Kaufkriterium.

Organizing the Content

- **Two-Panel Selector**
  Um einerseits eine Übersicht an Elementen und andererseits eine detailierte Ansicht eines Elementes zu bieten, sind Two-Panel Selectors eine gute Möglichkeit. Solche gesplitteten Frames werden beispielsweise oft von E-Mail-Clients verwendet.

- **Canvas Plus Palette**
  Wenn Benutzer Objekte auf einer Arbeitsfläche erstellen und bearbeiten möchten, bietet sich ein Canvas an. Die möglichen Objekte und Interaktionsmöglichkeiten mit diesen, werden durch eine Palette vorgegeben. Zum Einsatz kommen solche Canvases in allen gängigen Bildverarbeitungsprogrammen.

- **One-Window Drilldowns**
  One-Window Drilldowns umfassen die Funktion in einem einzigen Fenster. Ein gängiger Einsatz sind beispielsweise Handyapplikationen oder Browser. Solche One-Window Drilldowns sind simpel und klar strukturiert. Es gilt das Hier und Jetzt und nicht das was war oder sein wird.

- **Alternative Views**
  Gib dem Benutzer die Möglichkeit, eine andere Perspektive auf die Information zu bekommen. Dies bezieht sich nicht auf das Design, sondern auf die Struktur.

- **Wizards**
  Für lange und komplizierte Prozesse bietet es sich an, einen Leitfaden zu konstru-

ieren, der den Benutzer führt und ihm ermöglicht, auf die wesentlichen Merkmale eines Schrittes zu konzentrieren.

- **Extras on Demand**
  Die notwendigen Interaktionsmöglichkeiten sollten von Beginn an Angezeigt werden, zusätzliche Funktionen sollten On Demand hinzugeschaltet werden können.

- **Intriguing Branches**
  Benutzer sind von Natur aus neugierig. Gerade bei Internetseiten kommt es oft vor, dass Benutzer von ihrem eigentlichen Ziel abweichen und einem Link zu einem anderen interessanten Thema folgen.

- **Multi-Level Help**
  Eine Hilfe für Benutzer muss umfassend sein. Dies bedeutet unter anderem, dass sie auf verschiedenen Ebenen existiert. Es können Ballontipps, Hints oder auch ein ausführliche Online-Hilfe dazu beitragen, eine solche umfassende Hilfe zu erreichen.

Getting Around

- **Clear Entry Point**
  Es sollten nur wenige Eintrittpunkte zu Prozessen gegeben werden. Diese sollten aufgabenorientiert und gut beschrieben sein.

- **Global Navigation**
  Globale Navigationpunkte, wie etwa verschiedene Anwendungsbereiche einer Applikation, sollten durchweg verfügbar sein.

- **Hub and Spoke**
  Alle Kategorien sollten isoliert von den anderen sein. Sie sollen für sich allein stehen und nur durch die Hauptnavigation erreichbar sein.

- **Pyramid**
  Bei Interaktionsfolgen sollten vorher Übersichtspunkte angezeigt werden, bei denen ein Prozess beginnen kann, sofern nicht ein streng sequentieller Ablauf gefordert ist. Ein Beispiel wäre eine Übersichtsseite einer Internetpräsenz, die auf einzelne Artikel verweist, die untereinander mit „vor“ und „zurück“ sequentiell durchlaufen werden können.

- **Model Panel**
  Es sollten nur dann Interfaces ohne Navigationsmöglichkeiten angezeigt werden,

wenn auf eine notwendige Benutzereingabe gewartet werden muss. Eine häufige Anwendung ist ein „Speichern - Nicht Speichern - Abbrechen“-Dialog.

- **Sequence Map**
  Bei sequentiellen Abläufen, wie beispielsweise Wizards, sollte dem Benutzer mitgeteilt werden, in welchem Teil der Sequenz er sich gerade befindet.
- **Breadcumbs**
  Breadcumps zeigen dem Benutzer in welcher hirachischen Stufe er sich im Verhältnis zu Hauptseite befindet. Bekannt sind Breadcumbs von Internetseiten, aber auch dem Windows Explorer.
- **Annotated Scrollbar**
  Scrollbars sollten mit relevanten Informationen angereichert werden. Beispielweise könnten Seitenanzahl und Überschrift der aktuellen Seite beim Scrollen durch ein Office-Dokument angezeigt werden.
- **Color-Coded Sectons**
  Unterschiedlichen Kategorien einer Applikation können durch unterschiedliche Farbgebungen identifiziert werden.
- A**nimated Transition**
  Bewegung macht vieles natürlicher. So fühlt sich eine selbst steuerbare Navigation über eine Karte, wie beispielsweise Google Earth, natürlich an, als von Sektor zu Sektor navigieren zu müssen, wie es noch vor einigen Jahren war.
- **Escape Hatch**
  Einzelne Interfaces, die nur eingenschränkte Navigationsmöglichkeiten bieten, sollten klar erkennbare Austrittspunkte bieten.

Organizing the Page

- **Visual Framework**
  Alle Masken sollten ein einheitliches Design haben, jedoch sollte dieses Design so flexibel sein, dass es verschiedene Inhalte darstellen kann.
- **Center Stage**
  Der wichtigste Part des Interfaces sollte im größten Fenster oder Frame enthalten und die sekundären Interfaceelemente um den Hauptinhalt herum platziert sein.
- **Titled Sections**
  Themen sollten in Kategorien eingeteilt werden, welche einen Titel haben und sich visuell voneinander unterscheiden.

- **Card Stack**
  Kategorien sollten in getrennten Panels oder Reitern eingebettet sein.

- **Closable Panels**
  Panels sollten vom Benutzer geschlossen oder minimiert werden können, um für die aktuelle Aufgabe irrelevante Inhalte schließen zu können.

- **Movable Panels**
  Panals sollten frei verschiebar sein, um sich ein individuelles Layout schaffen zu könnsen.

- **Right/Left Alignment**
  Bei zweispaltigen Tabellen sollte die linke Spalte rechts und die rechte Spalte links ausgerichtet sein.

- **Diagonal Balance**
  Die Abstände an der oberen linken und der unteren rechten Ecke zu den Inhalten sollten gleich sein.

- **Property Sheet**
  Eine zweispaltige Tabelle oder ein Formularlayout zeigen dem Benutzer, dass Eigenschaften eines Objektes geändert werden können.

- **Responsive Disclosure**
  Das Interface sollte beim Start der Applikation minimalistisch sein und den Benutzer mit der Zeit in das System einführen und die Interaktionsmöglichkeiten erweitern.

- **Responsive Enabling**
  Ein Interface sollte mit deaktivierten Elementen starten, wenn diese Elemente eine vorausgehende Interaktion benötigen, um eine sinnvolle Benutzung gewährleisten werden können.

- **Liquid Layout**
  Wenn die Fenstergröße verändert wird, sollte sich das Layout automatisch an die Änderung anpassen.

Doing Things

- **Button Groups**
  Buttons die Interaktionsmöglichkeiten anbieten, die in einem engen Zusammenhang stehen, sollten in einer Gruppe präsentiert werden. Die Buttons sollten die

gleiche horizontale oder vertikale Ausrichtung haben und die Gruppe sollte nicht mehr als drei bis vier Buttons umfassen.

- **Action Panel**
  Wichtige Interaktionmöglichkeiten, oder solche die oft benötigt oder genutzt werden, sollten auf einem Action Panel zusammengefasst werden.

- **Prominent „Done“ Button**
  Es muss ein passend beschriebener Button vorhanden sein, der die Interaktion abschließt.

- **Smart Menu Items**
  Menüpunkte sollten klare Metaphern besitzen, die genau das beschreiben, was die Aktion dahinter bewirkt.

- **Preview**
  Es sollte eine Vorschau angezeigt werden, wie sich beispielsweise Inhalte ändern würden, wenn die Aktion durchgeführt werden würde.

- **Progress Indicator**
  Dem Benutzer soll gezeigt werden wie viel Zeit noch benötigt wird, um einen zeitlich begrenzten Prozess zu Ende zu bringen.

- **Cancelability**
  Dem Benutzer soll es möglich sein, Prozesse die zeitintesiv sind, beispielsweise das Laden von Bibliotheken, ohne ungewollte Nebenwirkungen abzubrechen.

- **Multi Level Undo**
  Es sollte eine einfache Möglichkeit für den Benutzer bestehen, ausgeführte Aktionen oder Änderungen rückgängig machen zu können.

- **Command History**
  Ausgeführte Kommandos sollen nachträglich verfügbar sein.

- **Macros**
  Der Benutzer soll verschiedene aufeinander folgende Aktionen zu Makros zusammenfassen können.

Showing Complex Data

- **Overview Plus Detail**
  Bei einer Grafik sollte eine detaillierte Ansicht und eine Übersichtskarte zur Navigation vorhanden sein.

- **Datatips**
  Wenn der Mauszeiger über ein Objekt bewegt wird, sollen die enthaltenen Daten in einem Tooltip oder Ähnlichem angezeigt werden.

- **Dynamic Queries**
  Suchergebnisse von Abfragen sollten nachträglich anpassbar sein, um die Ergebnismenge einzugrenzen oder anders darzustellen.

- **Data Brushing**
  Der Benutzer sollte die Möglichkeit haben, verschiedene Sichten auf eine Datenmenge zu bekommen.

- **Local Zooming**
  Wenn alle Daten in einer hohen Dichte angezeigt werden, sollte es möglich sein, einzelne Elemente zu vergrößern, um die Einzelheiten des Objektes anzeigen zu können. Die anderen Elemente treten in dem Moment in den Hintergrund.

- **Row Striping**
  Zwei aufeinander folgende Zeilen einer Tabelle sollten unterschiedliche Hintergrundfarben haben, um sich voneinander abzuheben.

- **Sortable Table**
  Tabellarische Daten sollten nach den Spaltennamen vom Benutzer sortiert werden können.

- **Jump to Item**
  Wenn ein Benutzer den Namen eines Objektes eingibt, sollte zu dem Objekt gesprungen und dieses selektiert werden.

- **New-Item Row**
  Das letzte Element in einer Tabelle sollte als Interaktionselement bestehen, dass es ermöglicht, einen neuen Datensatz einzufügen.

- **Cascading List**
  Hierachische Listen sollen aufeinander aufbauen und durch markierte Elemente das Elternelement anzeigen.

- **Tree Table**
  Hierachische Daten können in Tabellenform zusammengebracht und durch Einrückungen kann die Hierarchie identifiziert werden.

- **Multy-Y Graph**
  Mehrere Graphen können in einem Zusammengefasst werden, indem sie dieselbe X-Achse benutzen und in der Y-Achse übereinander stehen.

- **Small Mutliples**
  Mehrere Bilder von Daten in verschiedenen Dimensionen können einen Verlauf der Daten in verschiedenen Dimensionen anzeigen.

- **Treemap**
  Um mehrdimensionale und/oder hierarchische Daten darzustellen, können diese in Rechtecken in verschiedenen Größen dargestellt werden. Diese Rechtecke können wieder eingebettete Rechtecke enthalten, die durch Farben einen Hierarchie bilden. Verschiedene Gruppen werden durch Label identifiziert.

Getting Input from Users

- **Forgiving Format**
  Die Eingaben des Benutzer sollten hinsichtlich der Semantik überprüft und dementsprechend ausgewertet werden.

- **Structured Format**
  Es sollten mehrere Eingabefelder verwendet werden, wenn es der Struktur der einzugebenden Daten entspricht, z.B. mehrspaltige Serials.

- **Fill-in-the-Blanks**
  Eingabefelder, in die der Benutzer etwas eintragen soll, sollten leer sein.

- **Input Hints**
  Hints sollen erklären, welche Daten gefordert und wie diese einzugeben sind.

- **Input Prompt**
  Es soll ein Text im Eingabefeld stehen, der sagt, was eingefügt werden soll, oder eine Auswahlmöglichkeit mit einem Dropdown Menü gegeben werden.

- **Autocompletion**
  Bei der Eingabe durch den Benutzer sollen Möglichkeiten für die Vervollständigung angegeben werden.

- **Dropdown Chooser**
  Ein Menüelement kann durch ein Dropdown Menü erweitert werden, um auf einem Panel ein komplexeres Interface anzubieten.

- **Illustrated Choices**
  Es sollen Bilder statt Wörter verwendet werden, um Auswahlmöglichkeiten anzubieten.

- **List Builder**
  Es sollen sowohl die Quell- als auch die Zielliste angezeigt werden. Der Benutzer soll die Elemente zwischen diesen Listen tauschen können.

- **Good Defaults**
  Wo immer es angebracht ist, sollten Standardwerte in Eingabefeldern stehen, die am ehesten den Eingaben des Benutzers entsprechen.

- **Same-Page Error Messages**
  Fehlermeldungen sollen über dem Formular wo der Fehler entstanden ist, platziert werden. Eine angemessene Kennzeichnung sollte dann den Teil des Formulars beschreiben, wo der Fehler entstanden ist.

Builders and Editors

- **Edit-in-Place**
  Es soll ein kleiner dynamischer Editor verwendet werden, mit dem der Benutzer Änderungen an Ort und Stelle ändern kann, ohne ein weiteres Panel öffnen zu müssen.

- **Smart Selection**
  Die Software soll intelligent genug sein, um Gruppenzugehörigkeiten automatisch zu erkennen und zu selektieren.

- **Composite Selection**
  Verschiedene Mausgesten oder Mausklicks sollen an verschiedenen Positionen des Bildschirm unterschiedliche Interaktionen ermöglichen.

- **One-Off Mode**
  Wenn ein bestimmter Modus aktiviert wurde, soll nach Beendigung der Interaktion automatisch wieder in den Standardmodus gewechselt werden.

- **Spring-Loaded Mode**
  Der Benutzer soll durch Drücken einer Tastatur- oder Maustaste in einen Modus kommen, den er beim Loslassen der Taste wieder verlässt.

- **Constrained Resize**
  Bei der Änderung der Größe eines Objektes sollten verschiedene Bedürfnisse beachtet werden, zum Beispiel dass das Seitenverhältnis beibehalten wird.

- **Magnetism**
  Objekte sollen magnetisch sein, sich also an naheliegende Objekte anheften, wenn sie bewegt werden.

- **Guides**
  Vertikale und horizontale Linien sollen den Benutzer bei der Ausrichtung von Objekten unterstützen.

- **Paste Variations**
  Es sollen verschiedene Möglichkeiten des „Pasten“, neben der Standardoperation, unterstützt werden. Beispielsweise den Text einer Tabelle als Text oder als Tabellenobjekt zu kopieren.

Making it Look Good

- **Deep Background**
  Es soll ein Bild oder ein Gradient im Hintergrund stehen, von dem sich der Vordergrund visuell abhebt.

- **Few Hues, Many Values**
  Es sollen nicht mehr als drei Farbtöne verwendet werden. Die einzelnen Farbtöne können dann in verschieden hellen Ausprägungen versendet werden.

- **Corner Treatments**
  Anstatt der gewöhnlichen rechteckigen Ecken, sollen abgerundete verwendet werden.

- **Borders that Echo Fonts**
  Bei der Erstellung von Rahmen soll darauf geachtet werden, dass die Rahmenlinien dieselbe Dicke und Farbe wie einer der Hauptschriften haben.

- **Hairlines**
  Haarlinien sollen das Layout sinnvoll unterteilen.

- **Contrasting Font Weights**
  Schriften sollen in verschiedenen Farben und Stärken verwendet werden, um verschiedene Level der Information und somit auch Interesse beim Benutzer zu erwecken.

- **Skins**
  Neben dem Standardskin soll es Benutzern möglich sein, eigene Skins für die Applikation zu erstellen.

7.4.4 Evaluation und Testing

Heuristiken

Heuristiken können nicht sehr viele Usability-Probleme aufdecken, jedoch sind Fehler, die durch Heuristiken entdeckt werden, meist sehr gravierend. Heuristiken basieren auf Grundregeln die befolgt werden sollten, wenn man interaktive Systeme entwickelt. Heuristiken können nicht bestimmen ob beispielsweise der Maskenfluss aufgabenangemessen oder erwartungskonform ist, jedoch kann im Vorfeld abgeschätzt -allerdings nicht bestimmt!- werden, ob der cognitive load[8] zu hoch ist.
Heuristiken nach Nielsen und Molig:

- **Visibility of system status**
  Die Benutzer müssen jederzeit erkennen in welchem Vorgang sie sich gerade befinden und über aktuelle Geschehnisse informiert werden. Die Benutzer sollen beispielsweise nicht warten müssen, ohne dies angekündigt zu bekommen.

- **Match between system and the real world**
  Das System soll die Sprache der Benutzer sprechen und nicht die der Entwickler oder einer andere Domäne. Abläufe sollten logisch gegliedert sein und bekannten Konzepten folgen, um die Benutzer bestmöglich zu unterstützen.

- **User control and freedom**
  Benutzer sollen sich frei bewegen, aber Vorgänge auch abbrechen können, um beispielsweise Fehleingaben zu verwerfen.

- **Consistency and standards**
  Benutzer sollen sich nicht wundern müssen, warum verschiedene Wörter, Konzepte oder Metaphern bei gleichem Kontext vorhanden sind. Es soll sich einheitlich an Plattform-Konventionen gehalten werden.

- **Error prevention**
  Fehlermeldungen sind soweit zu vermeiden wie möglich, beispielsweise durch Plausibilitätsabfragen.

- **Recognition rather than recall**
  Die Gedächnisleistung der Benutzer soll so klein gehalten werden wie möglich (7

[8] Cognitive load beschreibt die Belastung des Arbeitsgedächtnisses mit Informationen die verarbeitet oder bereitgehalten werden müssen. Bei einem Maskenwechsel könnten das beispielsweise Informationen sein, die im Kopf behalten und in der nächsten Maske wieder eingegeben werden müssen. Um den cognitive load zu senken könnte man an dieser Stelle eine Möglichkeit einbauen, die Daten zu übernehmen.

± 2 Informationseinheiten). Nötige Informationen sollen jederzeit verfügbar oder vorher kopierbar sein.

- **Flexibility and efficiency of use**
  Erfahrene wie auch ungeübte Benutzer sollten angemessen unterstützt werden. Als Hilfestellung für erfahrene Benutzer sind beispielsweise Shortcuts eine gute Methode.

- **Aesthetic and minimalist design**
  Informationen sollten optimal aufbereitet und angemessen repräsentiert werden. Unnötige oder nur selten benötigte Informationen sind zu vermeiden.

- **Help users recognize, diagnose, and recover from errors**
  Fehlermeldungen sollten kurz und präzise sein und bei der Fehlerfindung helfen. Niemals sollten die Benutzer die Schuld für einen Fehler zugesprochen bekommen.

- **Help and documentation**
  Eine Dokumentation die an die Bedürfnisse der Benutzer angepasst ist, sollte vorhanden sein. Sie muss gut strukturiert und durchsuchbar sein.

Fragebögen

Fragebögen sind ein oft eingesetztes Mittel um die Softwarequalität zu prüfen. Fragebögen können formativ, während der Entwicklung, und summativ, nach Abschluss der Entwicklung, eingesetzt werden und lassen sich in qualitative, wobei der Benutzer[9] seine persönlichen Eindrücke selbst formuliert, und qualitative, wobei der Benutzer zwischen mehreren Aussagen wählen kann, unterscheiden. Einer der größten Vorteile ist, dass der Aufwand einen Fragenbogen auszufüllen relativ gering ist. Besonders bei quantitativen Fragebögen ist dies der Fall. In der Regel wird dies nicht mehr als 10 Minuten in Anspruch nehmen. Daher sind Fragebögen geeignet während und nach jeder Iteration als Evaluationsmethode eingesetzt zu werden. Man sollte darauf achten, dass Fragebögen von geschulten Personen[10] erstellt werden. Im Regelfall reicht es aus, auf vorgefertigte Fragebögen, wie den der ISO 9241-110[11], der die Grundsätze der Dialoggestaltung prüfen, oder SUMI[12] , zurück zu greifen.

[9] Anstatt des Benutzers kann auch ein Stellvertreter, der sich in der Domäne auskennt, die Befragung durchführen. Um die Gebrauchstauglichkeit prüfen zu können, sollten jedoch regelmäßig die Benutzer persönlich den Fragebogen ausfüllen.

[10] Als Vergleich für die Komplexität dieses Fachgebietes: Psychologiestudenten lernen in drei Semestern wie man Fragebögen erstellt und auswertet

[11] Einen an die ISO 9241 angelehnten Fragebogen findet sich beispielsweise im Datech Leitfaden für Usability [DAT08], S. 168

[12] Der SUMI (Software Usability Measurement Inventory) Fragebogen ist auf der Internetseite `http://sumi.ucc.ie/` (Sichtung: 16.07.2009) erhältlich

Die Software...	— – - +/- + ++ +++	Die Software...
ist kompliziert zu bedienen	++	ist unkompliziert zu bedienen

Tabelle 7.2: Beispiel für den Fragenbogen der 9241-110S

Gespräche

Eine umfangreiche Kommunikation zwischen den Anwendern, den Softwareentwicklern und den Usability Ingenieuren ist die Grundlage eines erfolgreichen[13] Projektes. Es ist zu empfehlen, die Gespräche persönlich zu führen, da einiges an Informationen verloren geht, wenn man telefonisch oder sogar per E-Mail kommuniziert. Gerade wenn sich Gesprächspartner nicht kennen, lässt sich die Distanz zwischen diesen nicht gut abbauen. Die Benutzer sollten nicht verhört werden, vielmehr sollen es lockere Gespräche sein, bei denen sich die Benutzer wohlfühlen und nicht denken, dass etwas, das sie sagen, falsch oder sogar lächerlich sein könnte. So sollte beispielsweise auch der Wunsch nach einer anderen Farbe erstmal erhört und als Anforderung des Benutzers aufgenommen werden. Wenn Benutzer Anforderungen formulieren ist es besonders wichtig, sie erklären zu lassen, wieso diese Anforderung wichtig ist und was damit erreicht werden soll. Da Benutzer den technischen Hintergrund oft nicht kennen sind ihnen alternative Lösungsmöglichkeit nicht bekannt, die Entwickler jedoch sind in der Lage, die Anforderung zu analysieren und die bestmögliche Lösung herauszuarbeiten. So kann aus der Anforderung „Ein Button, der ermöglicht, einen Benutzer aus der einen Datenbank in die andere Datenbank zu kopieren“ ein Batchjob werden, der einmal am Tag die Benutzer beider Datenbanken synchronisiert.

Cognitive Walkthrough

Der cognitive walkthrough (Lewis et al. 1990; Polsen et al. 1992a) ist eine Usability-Evaluations Methode, die den Fokus auf die Evaluation der Lernförderlichkeit durch die Exploration der Benutzer am Interface der Software hat (Vgl. [JN94]). Dieser Fokus ist dadurch motiviert, das beobachtet wurde, dass viele Benutzer durch Exploration den Umgang mit der Software lernen (Carrol and Rosson 1987, Fischer 1991). Bei einem cognitive walkthrough versetzt sich der Analyst in die Rolle des Benutzers, stellvertretend für diesen führt er die Analyse des Systems durch. Er benötigt detaillierte Informationen über Benutzer, Aufgabenbeschreibungen und Anforderungen, um ein valides Ergebnis erhalten zu können. Je detaillierter die Informationen, desto valider ist das Ergebnis.

[13] Erfolg wird hier durch die Ziele von Usability Engineering und die Ziele der Softwareentwicklung definiert

Die Herausforderung für den Analysten ist es, sich von seinem Grundverständnis und grundlegendem Handeln zu distanzieren und zu versuchen, aus der Perspektive der Benutzer das System zu betrachten.
Der cognitive walkthrough betrachtet die Lernförderlichkeit der Software, daraus ergeben sich folgenden Annahmen:

- Die Benutzer haben nur eine ungefähre Vorstellung des Systems
- Die Benutzer explorieren das System, um Funktionen zu prüfen und die Perfomanz zu steigern
- Die Benutzer wählen den Weg[14], der aus ihrer Sicht am schnellsten zum Ziel führt
- Die Benutzer interpretieren die Systemreaktion und mutmaßen ob die Reaktion zur Aufgabenerfüllung beiträgt
- Die Benutzer haben kein gesteigertes Interesse an formalem Training in dem interaktiven System

Anstelle eines einzelnen Analysten kann auch eine Gruppe, bestehend aus beispielsweise Designern oder Entwicklern, den cognitive walkthrough durchführen.

Labortests

Tests mit Benutzern oder deren Vertretern am aktiven System können in Usability-Labors durchgeführt werden. Der Nachteil an diese Methode ist, dass sich die Benutzer nicht in ihrer gewohnten Arbeitsumgebung befinden und dadurch der Nutzungskontext nicht der gleiche ist. Der Vorteil von Labortstests ist allerdings, dass die Testsysteme größtenteils vorbereitet sind. So sind Kameras installiert und Aufzeichnungssoftware auf den Systemen vorhanden, die die Aktionen (Mausbewegung, Tastatureingaben etc.) des Benutzers aufzeichnen. Solche Software kann auch analysieren, wie lange ein Proband auf eine Maske verweilt, bis er eine Aktion tätigt. Dadurch lässt sich beispielsweise erkennen, ob Benutzer die von ihnen gewünschte Funktion zeitnah finden. Neben der Analyse der Aktionen ist auch Eye-Tracking möglich, dadurch lässt sich analysieren wie ein Benutzer das Interface „liest“.
Neben dem Probanden muss ein Ansprechpartner für diesen im Raum sein. Der Ansprechpartner erklärt dem Benutzer die Aufgabe[15], weist ihn auf die Situation hin und was alles getestet wird. Er muss dem Probanden das Gefühl geben, dass nicht er, sondern das System getestet wird. Der Ansprechpartner kann Hilfestellung geben,

[14] Dies kann beispielsweise ein Icon auf einem Button sein. Ein Druckersymbol würde beispielsweise vermuten lassen, dass die aktuelle Ansicht gedruckt werden kann.
[15] Die Aufgabe sollte einen praxisnaher Anwendungsfall sein.

sollte dies aber nur in Ausnahmefällen tun, da, wenn ein Benutzer seine Aufgabe, aus systemtechnischen Gründen, nicht erfüllen kann, dies ein grober Fehler im Design ist, was festgehalten und nicht durch Hilfestellung übergangen werden sollte. Es ist möglich, Beobachter zuzulassen, diesen sollte jedoch nicht im selben, sondern in einem separaten Raum sitzen und den Test via Live-Übertragung verfolgen. Es ist nicht unüblich dass der Kunde, für den das System entwickelt wird, bei so einem Test als Beobachter anwesend ist. Zusätzlich können Personen aus dem Marketing oder der Entwicklung beiwohnen. Der Test sollte, auch wenn keine Zuschauer vorhanden sind, von einer Videokamera fest-

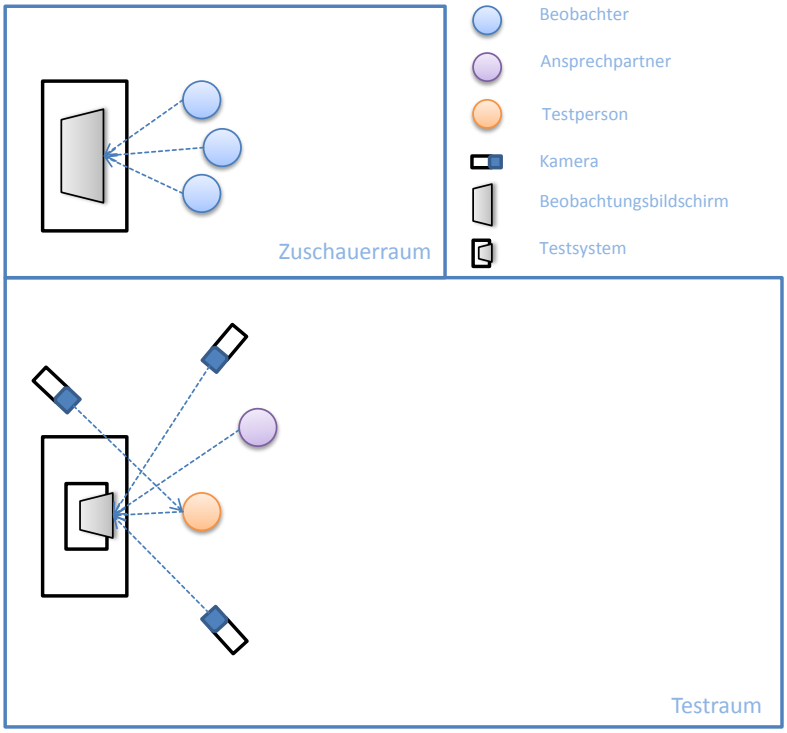

Abbildung 7.7: Beispielaufbau einer Testumgebung in einem Usability Labor

gehalten werden. Der Fokus liegt dabei auf der Interaktion des Benutzers. Zusätzlich kann auch eine zweite Kamera das Gesicht des Probanden aufnehmen, so ist es möglich, die Gesichtsausdrücke bei bestimmten Situation zu analysieren, was hilfreich ist, wenn man nach Stellen sucht, an dem der Tester Probleme hatte. Der Ansprechpartner kann sich jederzeit Notizen machen, sollte aber keine Analyse des Tests schreiben, dafür sind die Videoaufnahmen. Diese können nachträglich vom Entwicklungsteam gesichtet und ausgewertet werden.

Produktivtests

Bei Prdouktivtests liegt der Fokus auf dem Testen in der realen Arbeitsumgebung unter realen Umständen. Die Testaufgaben können, sofern dies unkritisch ist, reale Fälle sein. Die Gesamtsituation soll so praxisnah wie möglich sein, um den Benutzer in seiner gewohnten Umgebung beobachten zu können. So sollten auch typische Störfaktoren wie Druckergeräusche oder klingelnde Telefone nicht eliminiert werden. Wenn jedoch außergewöhnliche Umstände herrschen, wie etwa Bauarbeiten oder ähnliches, sollte der Test verschoben werden. Bei Produktivtests sollte in jedem Fall auch ein Usability Experte anwesend sein, der den Tester beobachtet und diesem bei Problemen zur Seite steht. Wichtig ist, dass der Tester beobachtet wird, um auch kleine Probleme zu erkennen. Viele Probleme, wie etwa ein falsch interpretiertes Symbol, werden nicht erwähnt, da das Problem oft bei sich selbst gesucht wird, wenn das System nicht erwartungskonform reagiert. Es besteht auch die Möglichkeit, den Test aufzuzeichnen, um ihn später im Detail analysieren zu können.

8 Fazit

Ziel war es, einen Usability Engineering Prozess mit einem etablierten Softwareentwicklungsprozess in einem Framework mit agilem Vorgehen zu integrieren. Wie eine vorangegangen Analyse gezeigt hat, besteht gerade im Bereich des Usability Engineerings Verbesserungsbedarf. Für die Integration wurden relevante Standards[1] ausgewählt und gezeigt, dass eine Integration über gemeinsame Prozesse oder Projekt-Dokumente möglich ist. Auf Basis dieser Erkenntnisse wurde ein Framework entwickelt, das agil ist und die relevanten Standards der Softwareentwicklung und des Usability Engineerings berücksichtigt und in dem eine Integration beider Domänen über gemeinsame Prozesee und/oder Projekt-Dokumente stattfinden kann. Aus Sicht der aktuellen Normen ist es vollständig und lässt Freiraum für zusätzliche Prozesse, Methoden, Techniken oder Projekt-Dokumente. Die Umsetzung des Vorgehensmodells im VModell XT Editor, BM Rational Method Composer oder MicroTOOL InStep konnte aus zeitlichen Gründen nicht umgesetzt werden. Dies hätte einerseits die Planung eines Projektes mit diesem Framework für Projektmanager vereinfacht und andererseits ermöglicht, die Konstistenz und die Vollständigkeit besser zu prüfen.
Die durchgeführte Qualität ist letztendlich vom Team, dessen Wissenstand und Motivation abhängig. Kritisch zu sehen ist die Beschreibung der einzelnen Methoden und Techniken, da es aus Zeitgründen nicht möglich war zu Prüfen, ob Non-Usability Experten mit den Beschreibungen ein korrektes Usability Engineering durchführen können. Beim Einsatz des Frameworks muss darauf also besonders geachtet werden.
Ob das Framework eingesetzt werden kann, ist stark von dem Projekt und dem Kunden abhängig, der dieses in Auftrag gibt. Der Kunde muss ebenso von einem Usability Engineerings Prozess überzeugt sein, wie die Entwickler, da die Unterstützung des Kunden, beispielsweise bei Tests, von elementarer Bedeutung ist. Neben dem Kunden müssen auch deren Betreuer aus der Akquise geschult werden, um werben und beraten zu können. Ihnen muss bekannt sein, zu was sich ein Kunde verpflichtet, wie viel mehr ein Usability Prozess kostet und welche Vorteile für den Kunden daraus entstehen.

[1] ISO 12207, ISO 13407, ISO/PAS 18152

9 Ausblick

Das Framework ist relativ grob granular und bietet viele Erweiterungsmöglichkeiten. Besonders kann der Methoden- und Technikkatalog erweitert werden. Ebenso können weitere Ziele für Softwareentwicklung oder Usability hinzugefügt werden, um jeweils „state of the art“ zu sein. So kann das Framework an das jeweilige Projekt angepasst und so der Entwicklungsprozess etabliert werden. Es können verschiedene Instanzen des Frameworks im Unternehmen etabliert werden, die für verschiedene Situationen geeignet sind. Durch stetige Evaluation werden diese Instanzen dann optimiert. Als Bewertungskriterium könnte das Bewertungsmodell der ISO 15504 [ISOc] dienen. Das Framework schließt auch nicht aus, dass Softwareentwickler und Usability Ingineure in zwei getrennten Teams arbeiten. So ist es beispielweise möglich, nach erfolgreicher Etablierung eines „einfachen“ Usability Engineerings, Usability Experten einzustellen, die die Usability Ziele des Frameworks erfüllen und gemäß dessen eng mit den Softwareentwickler zusammenarbeiten.

Abbildungsverzeichnis

2.1 Schematisches Modell des „mutli-store memory“ nach Atkinson und Shiffrin (Vgl. [Ben05], S. 356) 12
2.2 feature integration theory von Ann Treisman 15

3.1 Vorgehensmodell der ISO 13407 19
3.2 Das IFIP Modell besitzt zwei Organisationsschnittstellen 20

4.1 Mögliche Durchstoßpunkte zwischen der Softwareentwicklung und dem Usability Engineering 37

5.1 Ablauf eines Scrum Projektes (Zeichnung nach Vorlage von Roman Pichler (Vgl. [Pic08], S.7)) 39

7.1 Vorgehensmodell des Frameworks 48
7.2 Das Vorgehensmodell im Detail innerhalb eines Sprints mit den verschiedenen Aktivitäten von Softwareentwicklung und Usability Engineering . 49
7.3 Das Vorgehensmodell im Detail innerhalb eines Sprints mit den verschiedenen Projekt-Dokumenten von Softwareentwicklung und Usability Engineering 50
7.4 Beispiel-HTA einer Nachhilfestunde 59
7.5 HTA Sequenz ausformuliert 60
7.6 HTA Plansequenz 61
7.7 Beispielaufbau einer Testumgebung in einem Usability Labor 79

Tabellenverzeichnis

4.1 Integrationspunkte über identische Aktivitäten der Softwareentwicklung (ISO 12207) und des Usability Engineerings (ISO 13407) (Vgl. [Kar08], S.59) 36

7.1 Beispiel von einem essential use case von der User-Intention an einem Bankautomaten 62
7.2 Beispiel für den Fragenbogen der 9241-110S 77

Glossar

Anforderung ..... Erfordernis oder Erwartung, das oder die festgelegt, üblicherweise vorausgesetzt oder verpflichtend ist.

Arbeitsaufgabe ..... Die zur Zielsetzung erforderlichen Aktivitäten

Arbeitssystem ..... Ein System, das aus Benutzern, Arbeitsmitteln, Arbeitsaufgaben und der physischen wie sozialen Umgebung besteht, um bestimmte Ziele zu erreichen.

Aufgabenangemessenheit Ein interaktives System ist aufgabenangemessen, wenn es den Benutzer unterstützt, seine Arbeitsaufgabe zu erledigen, d. h., wenn Funktionalität und Dialog auf den charakteristischen Eigenschaften der Arbeitsaufgabe basieren, anstatt auf der zur Aufgabenerledigung eingesetzten Technologie.

Aufgabenmodell .... Die Zerlegung einer Kernaufgabe in sequenzielle Schritte folgerichtiger Teilaufgaben im Zuge der Erfüllung einer vollständigen Tätigkeit.

Benutzer ..... Person, die mit dem interaktiven System arbeitet

Benutzungsschnittstelle alle Bestandteile eines interaktiven Systems (Software oder Hardware), die Informationen und Steuerelemente zur Verfügung stellen, die für den Benutzer notwendig sind, um eine bestimmte Arbeitsaufgabe mit dem interaktiven System zu erledigen.

Dialog ..... Interaktion zwischen einem Benutzer und einem interaktiven System in Form einer Folge von Handlungen des Benutzers (Eingaben) und Antworten des interaktiven Systems (Ausgaben), um ein Ziel zu erreichen.

Effektivität ..... Die Genauigkeit und Vollständigkeit mit der Benutzer ein bestimmtes Ziel erreichen.

Effizienz ..... Der im Verhältnis zur Genauigkeit und Vollständigkeit eingesetzte Aufwand mit dem Benutzer ein bestimmtes Ziel

Erfordernis ..... Eine notwendige Voraussetzung, die es ermöglicht, den in einem Sachverhalt des Nutzungskontexts enthaltenen Zweck effizient zu erfüllen.

Erwartungskonformität Ein Dialog ist erwartungskonform, wenn er den aus dem Nutzungskontext heraus vorhersehbaren Benutzerbelangen sowie allgemein anerkannten Konventionen entspricht.

Fehlertoleranz ..... Ein Dialog ist fehlertolerant, wenn das beabsichtigte Arbeitsergebnis trotz erkennbar fehlerhafter Eingaben entweder mit keinem oder mit minimalem Korrekturaufwand seitens des Benutzers erreicht werden kann.

Gebrauchstauglichkeit Das Ausmaß, in dem ein Produkt durch bestimmte Benutzer in einem bestimmten Nutzungskontext genutzt werden kann, um bestimmte Ziele effektiv, effizient und zufriedenstellend zu erreichen.

Lernförderlichkeit ... Ein Dialog ist lernförderlich, wenn er den Benutzer beim Erlernen der Nutzung des interaktiven Systems unterstützt und anleitet.

Methode ..... Eine Methode wird verwendet um größere Ziele zu erreichen. Eine Methode kann aus verschiedenen Techniken bestehen, hat konkrete Vorraussetzungen und hat eine konkrete Zieldefinition.

Nutzungskontext ... Die Benutzer, Arbeitsaufgaben, Arbeitsmittel (Hardware, Software und Materialien) sowie die physische und soziale Umgebung, in der das Produkt genutzt wird.

Prototyp ..... Die Darstellung der Gesamtheit oder eines Teils eines Produkts oder Systems, welche, gegebenenfalls mit Einschränkung, für eine Beurteilung verwendet werden kann.

Selbstbeschreibungsfähigkeit Ein Dialog ist in dem Maße selbstbeschreibungsfähig, in dem für den Benutzer zu jeder Zeit offensichtlich ist, in welchem Dialog, an welcher Stelle im Dialog er sich befindet, welche Handlungen unternommen werden können und wie diese ausgeführt werden können.

Steuerbarkeit ..... Ein Dialog ist steuerbar, wenn der Benutzer in der Lage ist, den Dialogablauf zu starten sowie seine Richtung und Geschwindigkeit zu beeinflussen, bis das Ziel erreicht ist.

Technik ..... Eine Technik ist ein Werkzeug, was in einer strikten abfolge verwendet wird um kleinere Ziele zu erreichen.

Validierung ..... Bestätigung durch Bereitstellung eines objektiven Nachweises, dass die Anforderungen für einen spezifischen beabsichtigten Gebrauch oder eine spezifische beabsichtigte Anwendung erfüllt worden sind.

Verifikation ..... Bestätigung durch Bereitstellung eines objektiven Nachweises, dass festgelegte Anforderungen erfüllt worden sind.

Ziel ..... Ein angestrebtes Arbeitsergebnis.

Zufriedenstellung ... Freiheit von Beeinträchtigung und positive Einstellungen gegenüber der Nutzung des Produkts.

Literaturverzeichnis

[Ben05] Benyon, David et al.: *Designing Interactive Systems.* Addison Wesley, Essex, England, 2005. ISBN 0-321-11629-1.

[DAT08] DATech: Deutsche Akkreditierungsstelle Technik in der TGA GmbH: *DATech Leitfaden Usability*, 2008.

[ISOa] ISO: *DIN EN ISO 13407 Benutzer-orientierte Gestaltung interaktiver Systeme.* 1999.

[ISOb] ISO: *ISO/IEC 12207 Systems and software engineering – Software life cycle processes.* 1995.

[ISOc] ISO: *ISO/IEC 15504-2 Information technology – Process assessment – Part 2: Performing an assessment.* 1998.

[ISOd] ISO: *ISO/IEC 9241 Ergonomics of human-system interaction.* 1998.

[ISOe] ISO: *ISO/PAS 18152 Ergonomics of human-system interaction – Specification for the process assessment of human-system issues.* 2003.

[JN94] Jakob Nielsen, Robert L. Mack: *Usability Inspection Methods.* Katherine Schowalter, New York, USA, 1994. ISBN 0-471-01877-5.

[Kar08] Karsten, Nebe: *Integration von Usability Engineering und Software Engineering: Konformitäts- und Rahmenanforderungen zur Bewertung und Definition von Softwareentwicklungsprozessen.* 2008.

[Pic08] Pichler, Roman: *Scrum.* dpunkt.verlag, Heidelberg, 2008. ISBN 978-3-89864-478-5.

[Ren09] René, Rappenhöner: *Assessment von Unternehmensprozessen zur Softwareentwicklung hinsichtlich ihres Reifegrades, gebrauchstaugliche Systeme zu generieren.* 2009.

[RLS04] Robert L. Solso, Kimberly M. Maclin, Otto Maclin: *Cognitive Psychology.* Allyn and Bacon, USA, 2004. 0- 2054290-9-2.

[Tid05] Tidwell, Jennifer: *Designing Interfaces.* O'Reilly, USA, 1. Auflage, 2005. ISBN 0-596-00803-1.

Eidesstattliche Erklärung

Ich versichere, die von mir vorgelegte Arbeit selbständig verfasst zu haben.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben.

Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsgbehörde vorgelegen.

Gummersbach, 28. August 2009

René Rappenhöner